

**ADVPL**

## Conteúdo

1.Objetivos do Curso.....	5
2.MÓDULO 04: Desenvolvendo aplicações em ADVPL.....	6
4. Estrutura de um Programa ADVPL.....	9
Linhas de Programa .....	10
2.1 Áreas de um Programa ADVPL.....	12
5 Declaração e Atribuição de Variáveis .....	15
2.3 Declaração de variáveis .....	16
2.4 Escopo de variáveis.....	17
2.5 Entendendo a influência do escopo das variáveis .....	22
6 Regras adicionais da linguagem ADVPL .....	23
2.7 Pictures de formatação disponíveis.....	24
7 Programas de Atualização.....	25
2.8 Modelo1() ou AxCadastro().....	26
2.9 Mbrowse().....	29
2.9.1 AxFunctions() .....	36
2.9.2 FilBrowse() .....	38
2.9.3 EndFilBrw().....	38
2.9.4 PesqBrw().....	39
2.9.5 BrwLegenda () .....	39
2.10MarkBrowse() .....	43
2.10.1 Funções de Apoio.....	44
2.11Modelo2() .....	49
2.11.2 Estrutura de um programa utilizando a Modelo2() .....	57
2.11.3 Função Modelo2().....	72
2.12.1 Estrutura de um programa utilizando a Modelo3() .....	80
2.12.2 Função Modelo3().....	92
2.14.3 IndRegua().....	97
2.15Funções Auxiliares para Arquivos de Trabalho e Temporários .....	98
9 Relatórios não gráficos.....	102
Sintaxe: AJUSTASX1(cPerg, aPergs) .....	107
2.16.8 PutSX1().....	108
2.17Estrutura de Relatórios Baseados na SetPrint().....	110
10. Manipulação de arquivos I.....	122
11. Oficina de programação I.....	141
2.20.1 RptStatus() .....	145
INCREGUA().....	150
2.20.5 MsgRun().....	160
Sintaxe: .....	166

12. Conceitos de orientação à objetos .....	183
2.24Definições .....	183
Polimorfismo.....	185
2.25Conceitos Básicos.....	186
2.26O Modelo de Objetos (OMT) .....	188
2.26.3 Operações e Métodos.....	190
2.26.4 Sugestões de desenvolvimento .....	191
2.28Estrutura de uma classe de objetos em ADVPL.....	195
Utilização de funções em uma classe de objetos .....	197
Herança entre classes .....	198
Construtor para classes com herança.....	200
MÓDULO 06: ADVPL ORIENTADO À OBJETOS I .....	201
Classes da interface visual .....	203
MSCALEND().....	204
Documentação dos componentes da interface visual.....	208
2.30Particularidades dos componentes visuais.....	209
2.31Captura de informações simples (Multi-Gets).....	211
2.32Captura de múltiplas informações (Multi-Lines .....	217
2.32.1 MsGetDB().....	218
2.32.2 MsGetDados() .....	223
2.33Barras de botões.....	240
2.33.1 EnchoiceBar() .....	241
2.33.2 TBar().....	243
2.33.4 Imagens pré-definidas para as barras de botões.....	249
16. Arredondamento .....	250
17. Utilização de Identação.....	251
18. Capitulação de Palavras-Chave .....	252
2.34Palavras em maiúsculo .....	253
19. Utilização da Notação Húngara.....	254
20. Técnicas de programação eficiente .....	254
Criação de funções segundo a necessidade .....	255
Codificação auto-documentável.....	256
Criação de mensagens sistêmicas significantes e consistentes.....	263
GUIA DE REFERÊNCIA RÁPIDA: FUNÇÕES E COMANDOS ADVPL.....	267
Matemáticas .....	270
Análise de variáveis.....	274
Manipulação de arrays .....	275
Manipulação de blocos de código .....	286
Manipulação de strings.....	289

BITON()	292
Manipulação de variáveis numéricas	312
ALEATORIO()	313
RANDOMIZE()	314
Manipulação de arquivos	316
Função auxiliar: PARBOXFILE()	320
Função auxiliar: MARKFILE()	321
Função auxiliar: MARCAOK()	323
Manipulação de arquivos e índices temporários	348
Manipulação de bases de dados	350
Controle de numeração seqüencial	398
Controle de processamentos	419
AXDELETA()	474
Interfaces visuais para aplicações	475
Recursos das interfaces visuais	478

## **1. Objetivos do Curso**

### **Objetivos específicos do curso:**

Ao final do curso o treinando deverá ter desenvolvido os seguintes conceitos, habilidades e atitudes:

#### **a) Conceitos a serem aprendidos**

- estruturas para implementação de relatórios e programas de atualização
- estruturas para implementação de interfaces visuais
- princípios do desenvolvimento de aplicações orientadas a objetos

#### **b) Habilidades e técnicas a serem aprendidas**

- desenvolvimento de aplicações voltadas ao ERP Protheus
- análise de fontes de média complexidade
- desenvolvimento de aplicações básicas com orientação a objetos

#### **c) Atitudes a serem desenvolvidas**

- adquirir conhecimentos através da análise dos funcionalidades disponíveis no ERP Protheus;
- estudar a implementação de fontes com estruturas orientadas a objetos em ADVPL;
- embasar a realização de outros cursos relativos a linguagem ADVPL

## **2. MÓDULO 04: Desenvolvendo aplicações em ADVPL**

### **3. A linguagem ADVPL**

A Linguagem ADVPL teve seu início em 1994, sendo na verdade uma evolução na utilização de linguagens no padrão xBase pela Microsiga Software S.A. (Clipper, Visual Objects e depois FiveWin). Com a criação da tecnologia Protheus, era necessário criar uma linguagem que suportasse o padrão xBase para a manutenção de todo o código existente do sistema de ERP Siga Advanced. Foi, então, criada a linguagem chamada *Advanced Protheus Language*.

O ADVPL é uma extensão do padrão xBase de comandos e funções, operadores, estruturas de controle de fluxo e palavras reservadas, contando também com funções e comandos disponibilizados pela Microsiga que a torna uma linguagem completa para a criação de aplicações ERP prontas para a Internet. Também é uma linguagem orientada a objetos e eventos, permitindo ao programador desenvolver aplicações visuais e criar suas próprias classes de objetos.

Quando compilados, todos os arquivos de código tornam-se unidades de inteligência básicas, chamados *APO's* (de *Advanced Protheus Objects*). Tais *APO's* são mantidos em um repositório e carregados dinamicamente pelo PROTHEUS Server para a execução. Como não existe a linkedição, ou união física do código compilado a um determinado módulo ou aplicação, funções criadas em ADVPL podem ser executadas em qualquer ponto do ambiente Advanced Protheus.

O compilador e o interpretador da linguagem ADVPL é o próprio servidor PROTHEUS (PROTHEUS Server), e existe um ambiente visual para desenvolvimento integrado (PROTHEUSIDE) onde o código pode ser criado, compilado e depurado.

Os programas em ADVPL podem conter comandos ou funções de interface com o usuário. De acordo com tal característica, tais programas são subdivididos nas seguintes categorias:

#### **Programação Com Interface Própria com o Usuário**

Nesta categoria entram os programas desenvolvidos para serem executados através do terminal remoto do Protheus: o Protheus Remote. O Protheus Remote é a aplicação encarregada da interface e da interação com o usuário, sendo que todo o processamento do código em ADVPL, o acesso ao banco de dados e o gerenciamento de conexões é efetuado no Protheus Server. O Protheus Remote é o principal meio de acesso a execução de rotinas escritas em ADVPL no Protheus Server, e por isso permite executar qualquer tipo de código, tenha ele interface com o usuário ou não.

Porém, nesta categoria, são considerados apenas os programas que realizam algum tipo de interface remota utilizando o protocolo de comunicação do Protheus.

Podem-se criar rotinas para a customização do sistema ERP Microsiga Protheus, desde processos adicionais até relatórios. A grande vantagem é aproveitar todo o ambiente montado pelos módulos do ERP Microsiga Protheus. Porém, com o ADVPL, é possível até mesmo criar toda uma aplicação, ou módulo, desde o começo.

Todo o código do sistema ERP Microsiga Protheus é escrito em ADVPL.

## **Programação Sem Interface Própria com o Usuário**

---

As rotinas criadas sem interface são consideradas nesta categoria porque geralmente têm uma utilização mais específica do que um processo adicional ou um relatório novo. Tais rotinas não têm interface com o usuário através do Protheus Remote, e qualquer tentativa nesse sentido (como a criação de uma janela padrão) ocasionará uma exceção em tempo de execução. Estas rotinas são apenas processos, ou Jobs, executados no Protheus Server. Algumas vezes, a interface destas rotinas fica a cargo de aplicações externas, desenvolvidas em outras linguagens, que são responsáveis por iniciar os processos no servidor Protheus através dos meios disponíveis de integração e conectividade no Protheus.

De acordo com a utilização e com o meio de conectividade utilizado, estas rotinas são subcategorizadas assim:

### **Programação por Processos**

Rotinas escritas em ADVPL podem ser iniciadas como processos individuais (sem interface) no Protheus Server através de duas maneiras: Iniciadas por outra rotina ADVPL através da chamada de funções como StartJob() ou CallProc() ou iniciadas automaticamente na inicialização do Protheus Server (quando propriamente configurado).

### **Programação de RPC**

Através de uma biblioteca de funções disponível no Protheus (uma API de comunicação), podem-se executar rotinas escritas em ADVPL diretamente no Protheus Server, através de aplicações externas escritas em outras linguagens. Isto é o que se chama de *RPC* (de *Remote Procedure Call*, ou *Chamada de Procedimentos Remota*).

O servidor Protheus também pode executar rotinas em ADVPL em outros servidores Protheus através de conexão TCP/IP direta, utilizando o conceito de *RPC*. Do mesmo modo, aplicações externas podem requisitar a execução de rotinas escritas em ADVPL através de conexão TCP/IP direta.

## **Programação Web**

O Protheus Server pode também ser executado como um servidor Web, respondendo a requisições HTTP. No momento destas requisições, pode executar rotinas escritas em ADVPL como processos individuais, enviando o resultado das funções como retorno das requisições para o cliente HTTP (como por exemplo, um Browser de Internet). Qualquer rotina escrita em ADVPL que não contenha comandos de interface pode ser executada através de requisições HTTP. O Protheus permite a compilação de arquivos HTML contendo código ADVPL embutido. São os chamados arquivos ADVPL ASP, para a criação de páginas dinâmicas.

## **Programação TelNet**

TelNet é parte da gama de protocolos TCP/IP que permite a conexão a um computador remoto através de uma aplicação cliente deste protocolo. O PROTHEUS Server pode emular um terminal TelNet, através da execução de rotinas escritas em ADVPL. Ou seja, pode-se escrever rotinas ADVPL cuja interface final será um terminal TelNet ou um coletor de dados móvel.

## 4. Estrutura de um Programa ADVPL

Um programa de computador nada mais é do que um grupo de comandos logicamente dispostos com o objetivo de executar determinada tarefa. Esses comandos são gravados em um arquivo texto que é transformado em uma linguagem executável por um computador através de um processo chamado *compilação*. A compilação substitui os comandos de alto nível (que os humanos compreendem) por instruções de baixo nível (compreendida pelo sistema operacional em execução no computador). No caso do ADVPL, não é o sistema operacional de um computador que irá executar o código compilado, mas sim, o Protheus Server.

Dentro de um programa, os comandos e funções utilizados devem seguir regras de sintaxe da linguagem utilizada. Caso contrário, o programa será interrompido por erros. Os erros podem ser de compilação ou de execução.

Erros de compilação são aqueles encontrados na sintaxe que não permitem que o arquivo de código do programa seja compilado. Podem ser comandos especificados de forma errônea, utilização inválida de operadores, etc..

Erros de execução são aqueles que acontecem depois da compilação, quando o programa está sendo executado. Podem ocorrer por inúmeras razões, mas geralmente se referem às funções não existentes, ou à variáveis não criadas ou inicializadas, etc..

## **Linhas de Programa**

As linhas existentes dentro de um arquivo texto de código de programa podem ser linhas de comando, linhas de comentário ou linhas mistas.

### **Linhas de Comando**

Linhas de comando possuem os comandos ou instruções que serão executadas. Por exemplo:

```
Local nCnt  
Local nSoma := 0  
For nCnt := 1 To 10  
nSoma += nCnt  
Next nCnt
```

### **Linhas de Comentário**

Linhas de comentário possuem um texto qualquer, mas não são executadas. Servem apenas para documentação e para tornar mais fácil o entendimento do programa.

Existem três formas de se comentar linhas de texto. A primeira delas é utilizar o sinal de \* (asterisco) no começo da linha:

```
* Programa para cálculo do total  
* Autor: Microsiga Software S.A.  
* Data: 2 de outubro de 2001
```

Todas as linhas iniciadas com um sinal de asterisco são consideradas como comentário.

Pode-se utilizar a palavra NOTE ou dois símbolos da letra "e" comercial (&&) para realizar a função do sinal de asterisco. Porém, todas estas formas de comentário de linhas são obsoletas e existem apenas para compatibilização com o padrão xBase. A melhor maneira de comentar linhas em ADVPL é utilizar duas barras transversais:

```
// Programa para cálculo do total  
// Autor: Microsiga Software S.A.  
// Data: 2 de outubro de 2001
```

Outra forma de documentar textos é utilizar as barras transversais juntamente com o asterisco, podendo-se comentar todo um bloco de texto sem precisar comentar linha a linha:

```
/*  
Programa para cálculo do total  
Autor: Microsiga Software S.A.  
Data: 2 de outubro de 2001
```

```
*/
```

Todo o texto encontrado entre a abertura (indicada pelos caracteres /\*) e o fechamento (indicada pelos caracteres \*/) é considerado como comentário.

### **Linhas Mistas**

O ADVPL também permite que existam linhas de comando com comentário. Isto é possível adicionando-se as duas barras transversais (//) ao final da linha de comando e adicionando-se o texto do comentário:

```
Local nCnt
Local nSoma := 0 // Inicializa a variável com zero para a soma
For nCnt := 1 To 10
nSoma += nCnt
Next nCnt
```

### **Tamanho da Linha**

Assim como a linha física, delimitada pela quantidade de caracteres que pode ser digitado no editor de textos utilizado, existe uma linha considerada linha lógica. A linha lógica, é aquela considerada para a compilação como uma única linha de comando.

A princípio, cada linha digitada no arquivo texto é diferenciada após o pressionamento da tecla <Enter>. Ou seja, a linha lógica, é a linha física no arquivo. Porém, algumas vezes, por limitação física do editor de texto ou por estética, pode-se "quebrar" a linha lógica em mais de uma linha física no arquivo texto. Isto é efetuado utilizando-se o sinal de ponto-e-vírgula (;).

```
If !Empty(cNome) .And. !Empty(cEnd) .And. ; <enter>
!Empty(cTel) .And. !Empty(cFax) .And. ; <enter>
!Empty(cEmail)

GravaDados(cNome,cEnd,cTel,cFax,cEmail)

Endif
```

Neste exemplo existe uma linha de comando para a checagem das variáveis utilizadas. Como a linha torna-se muito grande, pode-se dividí-la em mais de uma linha física utilizando o sinal de ponto-e-vírgula. Se um sinal de ponto-e-vírgula for esquecido nas duas primeiras linhas, durante a execução do programa ocorrerá um erro, pois a segunda linha física será considerada como uma segunda linha de comando na compilação. E durante a execução esta linha não terá sentido.

## 2.1 Áreas de um Programa ADVPL

Apesar de não ser uma linguagem de padrões rígidos com relação à estrutura do programa, é importante identificar algumas de suas partes. Considere o programa de exemplo abaixo:

```
#include protheus.ch

/*
+=====
| Programa: Cálculo do Fatorial      |
| Autor   : Microsiga Software S.A.   |
| Data    : 02 de outubro de 2001       |
+=====

*/
User Function CalcFator()

Local nCnt
Local nResultado := 1 // Resultado do fatorial
Local nFator      := 5 // Número para o cálculo

// Cálculo do fatorial
For nCnt := nFator To 1 Step -1
nResultado *= nCnt
Next nCnt

// Exibe o resultado na tela, através da função alert
Alert("O fatorial de " + cValToChar(nFator) +
      " é " + cValToChar(nResultado))

// Termina o programa
Return
```

A estrutura de um programa ADVPL é composta pelas seguintes áreas:

- Área de Identificação
  - Declaração dos includes
  - Declaração da função
  - Identificação do programa

- Área de Ajustes Iniciais
  - Declaração das variáveis

- Corpo do Programa
  - Preparação para o processamento

## Processamento

### Área de Encerramento

## Área de Identificação

---

Esta é uma área que não é obrigatória e é dedicada à documentação do programa. Quando existente, contém apenas comentários explicando a sua finalidade, data de criação, autor, etc., e aparece no começo do programa, antes de qualquer linha de comando.

O formato para esta área não é definido. Pode-se colocar qualquer tipo de informação desejada e escolher a formatação apropriada.

```
#include "protheus.ch"

/*
+=====+
| Programa: Cálculo do Fatorial      |
| Autor   : Microsiga Software S.A.   |
| Data    : 02 de outubro de 2001       |
+=====+
*/

User Function CalcFator()
```

Opcionalmente podem-se incluir definições de constantes utilizadas no programa ou inclusão de arquivos de cabeçalho nesta área.

## Área de Ajustes Iniciais

---

Nesta área geralmente se fazem os ajustes iniciais, importantes para o correto funcionamento do programa. Entre os ajustes se encontram declarações de variáveis, inicializações, abertura de arquivos, etc.. Apesar do ADVPL não ser uma linguagem rígida e as variáveis poderem ser declaradas em qualquer lugar do programa, é aconselhável fazê-lo nesta área visando tornar o código mais legível e facilitar a identificação de variáveis não utilizadas.

```
Local nCnt
Local nResultado := 0 // Resultado do fatorial
Local nFator     := 10 // Número para o cálculo
```

## **Corpo do Programa**

---

É nesta área que se encontram as linhas de código do programa. É onde se realiza a tarefa necessária através da organização lógica destas linhas de comando. Espera-se que as linhas de comando estejam organizadas de tal modo que no final desta área o resultado esperado seja obtido, seja ele armazenado em um arquivo ou em variáveis de memória, pronto para ser exibido ao usuário através de um relatório ou na tela.

```
// Cálculo do fatorial  
For nCnt := nFator To 1 Step -1  
    nResultado *= nCnt  
Next nCnt
```

A preparação para o processamento é formada pelo conjunto de validações e processamentos necessários antes da realização do processamento em si.

Avaliando o processamento do cálculo do fatorial descrito anteriormente, pode-se definir que a validação inicial a ser realizada é o conteúdo da variável nFator, pois a mesma determinará a correta execução do código.

```
// Cálculo do fatorial  
nFator := GetFator()  
// GetFator – função ilustrativa na qual a variável recebe a informação do usuário.  
  
If nFator <= 0  
    Alert("Informação inválida")  
Return  
Endif  
  
For nCnt := nFator To 1 Step -1  
    nResultado *= nCnt  
Next nCnt
```

## **Área de Encerramento**

---

É nesta área que as finalizações são efetuadas. É onde os arquivos abertos são fechados, e o resultado da execução do programa é utilizado. Pode-se exibir o resultado armazenado em uma variável ou em um arquivo, ou, simplesmente, finalizar; caso a tarefa já tenha sido toda completada no corpo do programa. É nesta área que se encontra o encerramento do programa. Todo programa em ADVPL deve sempre terminar com a palavra chave return.

```

// Exibe o resultado na tela, através da função alert
Alert("O fatorial de " + cValToChar(nFator) +
      " é " + cValToChar(nResultado))

// Termina o programa
Return

```

## **5. Declaração e Atribuição de Variáveis**

### **2.2      Tipo de Dados**

O ADVPL não é uma linguagem de tipos rígidos (strongly typed), o que significa que variáveis de memória podem receber diferentes tipos de dados durante a execução do programa.

As variáveis podem também conter objetos, mas os tipos primários da linguagem são:

#### **Numérico**

O ADVPL não diferencia valores inteiros de valores com ponto flutuante, portanto podem-se criar variáveis numéricas com qualquer valor dentro do intervalo permitido.  
Os seguintes elementos são do tipo de dado numérico:

2
43.53
0.5
0.00001
1000000

Uma variável do tipo de dado numérico pode conter um número de dezoito dígitos incluindo o ponto flutuante, no intervalo de 2.2250738585072014 E-308 até 1.7976931348623158 E+308.

---

#### **Lógico**

Valores lógicos em ADVPL são identificados através de .T. ou .Y. para verdadeiro e .F. ou .N. para falso (independentemente se os caracteres estiverem em maiúsculo ou minúsculo).

## **Caractere**

---

Strings ou cadeias de caracteres são identificadas em ADVPL por blocos de texto entre aspas duplas ("") ou aspas simples (''):

```
"Olá mundo!"  
'Esta é uma string'  
"Esta é 'outra' string"
```

Uma variável do tipo caractere pode conter strings com no máximo 1 MB, ou seja, 1048576 caracteres.

## **Data**

---

O ADVPL tem um tipo de dados específico para datas. Internamente as variáveis deste tipo de dado são armazenadas como um número correspondente à **data Juliana**.

Variáveis do tipo de dados “Data” não podem ser declaradas diretamente, e sim através da utilização de funções específicas, como, por exemplo, CTOD() que converte uma string para data.

## **Array**

---

O Array é um tipo de dado especial. É a disposição de outros elementos em colunas e linhas. O ADVPL suporta arrays unidimensionais (vetores) ou multidimensionais (matrizes). Os elementos de um array são acessados através de índices numéricos iniciados em 1, identificando a linha e a coluna para quantas dimensões existirem.

Arrays devem ser utilizadas com cautela, pois se forem muito grandes podem exaurir a memória do servidor.

## **Bloco de Código**

---

O bloco de código é um tipo de dado especial. É utilizado para armazenar instruções escritas em ADVPL que poderão ser executadas posteriormente.

## **2.3 Declaração de variáveis**

Variáveis de memória são um dos recursos mais importantes de uma linguagem. São áreas de memória criadas para armazenar informações utilizadas por um programa para a execução de tarefas. Por exemplo, quando o usuário digita uma informação qualquer, como o nome de um produto, em uma tela de um programa esta informação é armazenada em uma variável de memória para posteriormente ser gravada ou impressa.

A partir do momento que uma variável é criada, deixa de ser necessário se referenciar ao seu conteúdo, sendo o nome a referência principal.

O nome de uma variável é um identificador único o qual deve respeitar um **máximo de 10 caracteres**. O ADVPL não impede a criação de uma variável de memória cujo nome contenha mais de 10 caracteres. **Porém, apenas os 10 primeiros serão considerados** para a localização do conteúdo armazenado.

Portanto, se forem criadas duas variáveis cujos 10 primeiros caracteres forem iguais, como nTotalGeralAnual e nTotalGeralMensal, as referências a qualquer uma delas no programa resultarão o mesmo, ou seja, serão a mesma variável:

```
nTotalGeralMensal := 100  
nTotalGeralAnual := 300  
Alert("Valor mensal: " + cValToChar(nTotalGeralMensal))
```

Quando o conteúdo da variável nTotalGeralMensal é exibido, o seu valor será de 300.

Isso acontece porque, no momento que esse valor foi atribuído à variável nTotalGeralAnual, o ADVPL considerou apenas os 10 primeiros caracteres (assim como o faz quando deve exibir o valor da variável nTotalGeralMensal), ou seja, considerou-as como a mesma variável. Assim o valor original de 100 foi substituído pelo de 300.

## 2.4 Escopo de variáveis

O ADVPL não é uma linguagem de tipos rígidos para variáveis, ou seja, não é necessário informar o tipo de dados que determinada variável irá conter no momento de sua declaração, e o seu valor pode mudar durante a execução do programa.

Também não há necessidade de declarar variáveis em uma seção específica do seu código fonte, embora seja aconselhável declarar todas as variáveis necessárias no começo, tornando a manutenção mais fácil e evitando a declaração de variáveis desnecessárias.

Para declarar uma variável deve-se utilizar um *identificador de escopo*, que é uma palavra chave que indica a que contexto do programa a variável declarada pertence. O contexto de variáveis pode ser *local* (visualizadas apenas dentro do programa atual), *público* (visualizadas por qualquer outro programa), entre outros.

### **O Contexto de Variáveis dentro de um Programa**

As variáveis declaradas em um programa ou função, são visíveis de acordo com o escopo onde são definidas. Como, também, do escopo depende o tempo de existência das variáveis. A definição do escopo de uma variável é efetuada no momento de sua declaração.

```
Local nNumero := 10
```

Os identificadores de escopo são:

- . Local
- . Static
- . Private
- . Public

O ADVPL não é rígido em relação à declaração de variáveis no começo do programa. A inclusão de um identificador de escopo não é necessária para a declaração de uma variável, contanto que um valor lhe seja atribuído.

```
nNumero2 := 15
```

Quando um valor é atribuído à uma variável em um programa ou função, o ADVPL criará a variável caso ela não tenha sido declarada anteriormente. A variável então é criada como se tivesse sido declarada como Private.

Devido a essa característica, quando se pretende fazer uma atribuição a uma variável declarada previamente, mas, escreve-se o nome da variável de forma incorreta, o ADVPL não gerará nenhum erro de compilação ou de execução. Pois, compreenderá o nome da variável escrito de forma incorreta como se fosse a criação de uma nova variável. Isto alterará a lógica do programa, e é um erro muitas vezes difícil de identificar.

### Variáveis de escopo local

---

Variáveis de escopo local são pertencentes apenas ao escopo da função onde foram declaradas e devem ser explicitamente declaradas com o identificador LOCAL, como no exemplo:

#### Function Pai()

```
Local nVar := 10, aMatriz := {0,1,2,3}

.

<comandos>

.

Filha()

.

<mais comandos>

.

Return(.T.)
```

Neste exemplo, a variável nVar foi declarada como local e atribuída com o valor 10. Quando a função Filha é executada, nVar ainda existe mas não pode ser acessada.

Quando a execução da função Pai terminar, a variável nVar é destruída. Qualquer variável com o mesmo nome no programa que chamou a função Pai não é afetada.

Variáveis de escopo local são criadas automaticamente cada vez que a função onde forem declaradas for ativada. Elas continuam a existir e mantêm seu valor até o fim da ativação da função (ou seja, até que a função retorne o controle para o código que a executou). Se uma função é chamada recursivamente (por exemplo, chama-se a si mesma), cada chamada em recursão cria um novo conjunto de variáveis locais.

A visibilidade de variáveis de escopo locais é idêntica ao escopo de sua declaração, ou seja, a variável é visível em qualquer lugar do código fonte em que foi declarada. Se uma função é chamada recursivamente, apenas as variáveis de escopo local, criadas na mais recente ativação, são visíveis.

### **Variáveis de escopo static**

---

Variáveis de escopo static funcionam basicamente como as variáveis de escopo local.

Mas, mantêm seu valor através da execução e devem ser declaradas explicitamente no código com o identificador STATIC.

O escopo das variáveis static depende de onde são declaradas. Se forem declaradas dentro do corpo de uma função ou procedimento, seu escopo será limitado àquela rotina. Se forem declaradas fora do corpo de qualquer rotina, seu escopo afeta a todas as funções declaradas no fonte. Neste exemplo, a variável nVar é declarada como static e inicializada com o valor 10:

#### **Function Pai()**

Static nVar := 10

.

<comandos>

.

Filha()

.

<mais comandos>

.

**Return(T.)**

Quando a função Filha é executada, nVar ainda existe mas não pode ser acessada.

Diferente de variáveis declaradas como LOCAL ou PRIVATE, nVar continua a existir e mantém seu valor atual quando a execução da função Pai termina. Entretanto, somente pode ser acessada por execuções subsequentes da função Pai.

## Variáveis de escopo private

---

A declaração é opcional para variáveis privadas. Mas podem ser declaradas explicitamente com o identificador PRIVATE.

Adicionalmente, a atribuição de valor a uma variável não criada anteriormente automaticamente cria a variável como privada. Uma vez criada, uma variável privada continua a existir e mantém seu valor até que o programa ou função onde foi criada termine (ou seja, até que a função onde foi criada retorne para o código que a executou). Neste momento, é automaticamente destruída.

É possível criar uma nova variável privada com o mesmo nome de uma variável já existente. Entretanto, a nova (duplicada) variável pode apenas ser criada em um nível de ativação inferior ao nível onde a variável foi declarada pela primeira vez (ou seja, apenas em uma função chamada pela função onde a variável já havia sido criada). A nova variável privada irá *esconder* qualquer outra variável privada ou pública (veja a documentação sobre variáveis públicas) com o mesmo nome enquanto existir.

Uma vez criada, uma variável privada é visível em todo o programa, enquanto não for destruída automaticamente. Quando a rotina que a criou terminar ou uma outra variável privada com o mesmo nome for criada em uma subfunção chamada (neste caso, a variável existente torna-se inacessível até que a nova variável privada seja destruída).

Em termos mais simples, uma variável privada é visível dentro da função de criação e todas as funções chamadas por esta, a menos que uma função chamada crie sua própria variável privada com o mesmo nome.

*Por exemplo:*

**Function Pai()**

Private nVar := 10

<comandos>

.

Filha()

<mais comandos>

.

**Return(T.)**

Neste exemplo, a variável nVar é criada com escopo private e inicializada com o valor 10. Quando a função Filha é executada, nVar ainda existe e, diferentemente de uma variável de escopo local, pode ser acessada pela função Filha. Quando a função Pai terminar, nVar será destruída e qualquer declaração de nVar anterior se tornará acessível novamente.



Importante

---

No ambiente ERP Protheus, existe uma convenção adicional a qual deve ser respeitada que variáveis em uso pela aplicação não sejam incorretamente manipuladas. Por esta convenção deve ser adicionado o caractere “\_” antes do nome de variáveis PRIVATE e PUBLIC. Maiores informações avaliar o tópico: Boas Práticas de Programação.

---

**Exemplo:** Private \_dData

---

### Variáveis de escopo public

---

Podem-se criar variáveis de escopo public dinamicamente no código com o identificador PUBLIC. As variáveis deste escopo continuam a existir e mantêm seu valor até o fim da execução da thread (conexão).

É possível criar uma variável de escopo private com o mesmo nome de uma variável de escopo public existente. Entretanto, não é permitido criar uma variável de escopo public com o mesmo nome de uma variável de escopo private existente.

Uma vez criada, uma variável de escopo public é visível em todo o programa onde foi declarada até que seja *escondida* por uma variável de escopo private criada com o mesmo nome. A nova variável de escopo private criada mascara a variável de escopo public existente, e esta se tornará inacessível até que a nova variável private seja destruída.

**Por exemplo:**

**Function Pai()**

Public nVar := 10

<comandos>

.

Filha()

<mais comandos>

.

**Return(T.)**

Neste exemplo, nVar é criada como public e inicializada com o valor 10. Quando a função Filha é executada, nVar ainda existe e pode ser acessada. Diferentemente de variáveis locais ou privates, nVar ainda existe após o término da execução da função Pai.

Diferentemente dos outros identificadores de escopo, quando uma variável é declarada como pública sem ser inicializada, o valor assumido é falso (.F.) e não nulo (nil).



No ambiente ERP Protheus, existe uma convenção adicional deve ser respeitada que determina que variáveis em uso pela aplicação não sejam incorretamente manipuladas. Por esta convenção, deve ser adicionado o caracter “\_” antes do nome de variáveis PRIVATE e PUBLIC. Para mais detalhes, veja o tópico: Boas Práticas de Programação.

**Exemplo:** Public \_cRotina

## **2.5 Entendendo a influência do escopo das variáveis**

Considere as linhas de código de exemplo:

```
nResultado := 250 * (1 + (nPercentual / 100))
```

Se esta linha for executada em um programa ADVPL, ocorrerá um erro de execução com a mensagem "variable does not exist: nPercentual", pois esta variável está sendo utilizada em uma expressão de cálculo sem ter sido declarada. Para solucionar este erro, deve-se declarar a variável previamente:

```
Local nPercentual, nResultado  
nResultado := 250 * (1 + (nPercentual / 100))
```

Neste exemplo, as variáveis são declaradas previamente utilizando o identificador de escopo *local*. Quando a linha de cálculo for executada, o erro de variável não existente, não mais ocorrerá. Porém, variáveis não inicializadas têm sempre o valor *default* nulo (Nil) e este valor não pode ser utilizado em um cálculo, pois também gerará erros de execução (nulo não pode ser dividido por 100). A resolução deste problema é efetuada inicializando-se a variável através de uma das formas:

```
Local nPercentual, nResultado  
nPercentual := 10  
nResultado := 250 * (1 + (nPercentual / 100))  
  
ou  
  
Local nPercentual := 10, nResultado  
nResultado := 250 * (1 + (nPercentual / 100))
```

A diferença entre o último exemplo e os dois anteriores é que a variável é inicializada no momento da declaração. Em ambos os exemplos, a variável é primeiro declarada e então inicializada em uma outra linha de código.

É aconselhável optar pelo operador de atribuição composto de dois pontos e sinal de igual, pois o operador de atribuição utilizando somente o sinal de igual pode ser facilmente confundido com o operador relacional (para comparação) durante a criação do programa.

## **6. Regras adicionais da linguagem ADVPL**

### **2.6 Palavras reservadas**

AADD	DTOS	INKEY	REPLICATE	VAL
ABS	ELSE	INT	RLOCK	VALTYPE
ASC	ELSEIF	LASTREC	ROUND	WHILE
AT	EMPTY	LEN	ROW	WORD
BOF	ENDCASE	LOCK	RTRIM	YEAR
BREAK	ENDDO	LOG	SECONDS	CDO
ENDIF	LOWER	SELECT	CHR	EOF
LTRIM	SETPOS	CMONTH	EXP	MAX
SPACE	COL	FCOUNT	MIN	SQRT
CTOD	FIELDNAME	MONT	STR	DATE
FILE	PCOL	SUBSTR	DAY	FLOCK
PCOUNT	TIME	DELETED	FOUND	PROCEDURE
TRANSFORM	DEVPOS	FUNCTION	PROW	TRIM
DOW	IF	RECCOUNT	TYPE	DTOC
IIF	RECNO	UPPER	TRY	AS
CATCH	THROW			

Palavras reservadas não podem ser utilizadas para variáveis, procedimentos ou funções;

Funções reservadas são pertencentes ao compilador e não podem ser redefinidas por uma aplicação;



*Importante*

Todos os identificadores que começarem com dois ou mais caracteres “\_” são utilizados como identificadores internos e são reservados.

Identificadores de escopo PRIVATE ou PUBLIC utilizados em aplicações específicas desenvolvida por clientes ou para clientes devem ter sua identificação iniciada por um caractere.

## **2.7 Pictures de formatação disponíveis**

Com base na documentação disponível no TDN, a linguagem ADVPL e a aplicação ERP Protheus admitem as seguintes pictures:

### **Dicionário de Dados (SX3) e GET**

---

<b>Funções</b>	
<b>Conteúdo</b>	<b>Funcionalidade</b>
A	Permite apenas caracteres alfabéticos.
C	Exibe CR depois de números positivos.
E	Exibe numérico com o ponto e vírgula invertidos (formato Europeu).
R	Insere caracteres diferentes dos caracteres de template na exibição, mas não os insere na variável do GET.
S<n>	Permite rolamento horizontal do texto dentro do GET, <n> é um número inteiro que identifica o tamanho da região.
X	Exibe DB depois de números negativos.
Z	Exibe zeros como brancos.
(	Exibe números negativos entre parênteses com os espaços em branco iniciais.
)	Exibe números negativos entre parênteses sem os espaços em branco iniciais.
!	Converte caracteres alfabéticos para maiúsculo.

<b>Templates</b>	
<b>Conteúdo</b>	<b>Funcionalidade</b>
X	Permite qualquer caractere.
9	Permite apenas dígitos para qualquer tipo de dado, incluindo o sinal para numéricos.
#	Permite dígitos, sinais e espaços em branco para qualquer tipo de dado.
!	Converte caracteres alfabéticos para maiúsculo.
*	Exibe um asterisco no lugar dos espaços em branco iniciais em números.
.	Exibe o ponto decimal.
,	Exibe a posição do milhar.

Exemplo 01 – Picture campo numérico

CT2\_VALOR – Numérico – 17,2  
Picture: @E 99,999,999,999,999.99

Exemplo 02 – Picture campo texto, com digitação apenas em caixa alta

A1\_NOME – Caracter - 40  
Picture: @!

## **7. Programas de Atualização**

Os programas de atualização de cadastros e digitação de movimentos seguem um padrão que se apóia no Dicionário de Dados.

Basicamente são três os modelos mais utilizados:

**Modelo 1 ou AxCadastro:**

Para cadastramentos em tela cheia. Exemplo: Cadastro de Cliente.

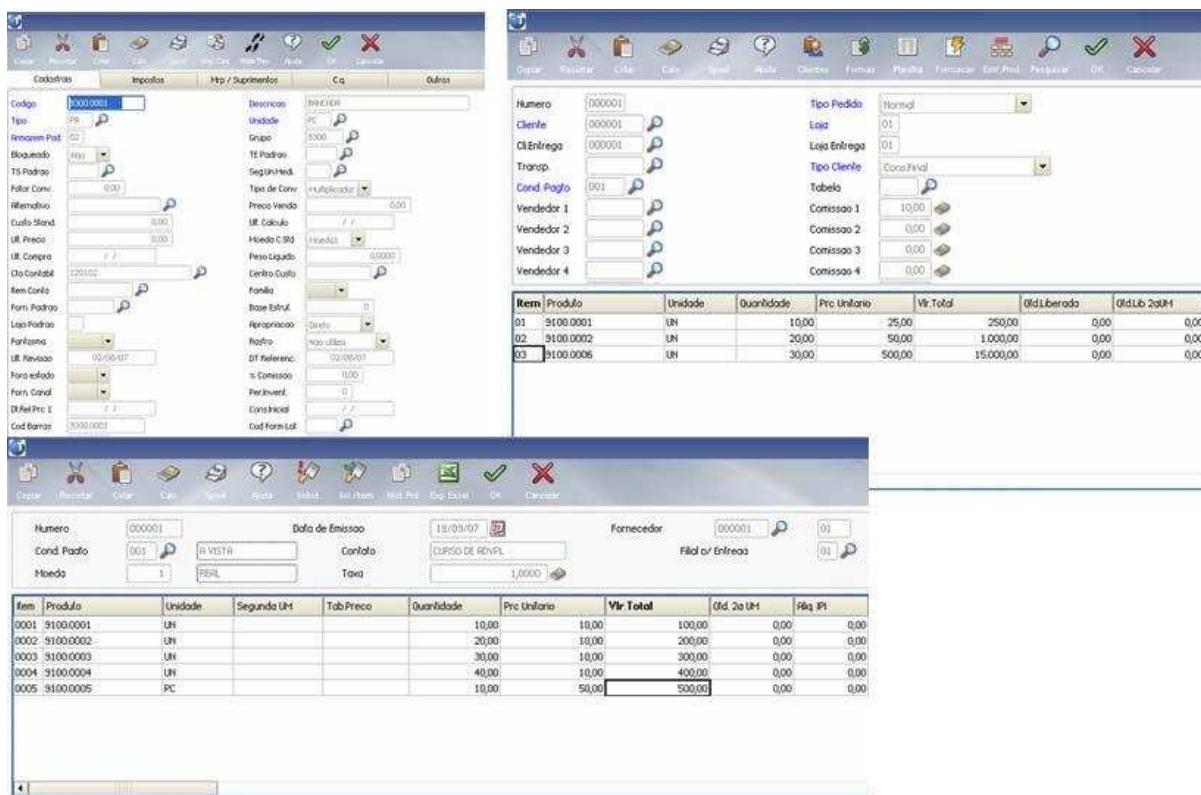
**Modelo 2:**

Cadastramentos envolvendo apenas uma tabela, mas com um cabeçalho e, opcionalmente, um rodapé e um corpo com quantidade ilimitada de linhas. Ideal para casos em que há dados que se repetem por vários itens e que, por isso, são colocados no cabeçalho. Exemplo: Pedido de Compra.

**Modelo 3:**

Cadastramentos envolvendo duas tabelas, um com dados de cabeçalho e outro digitado em linhas com os itens. Exemplo: Pedido de Vendas, Orçamento etc.

Todos os modelos são genéricos, ou seja, o programa independe da tabela a ser tratada, bastando praticamente que se informe apenas o seu Alias. O resto é obtido do Dicionário de Dados (SX3).



## 2.8 Modelo1() ou AxCadastro()

O AxCadastro() é uma funcionalidade de cadastro simples, com poucas opções de customização, a qual é composta de:

- Browse padrão para visualização das informações da base de dados, de acordo com as configurações do SX3 – Dicionário de Dados (campo browse).
- Funções de pesquisa, visualização, inclusão, alteração e exclusão padrões para visualização de registros simples, sem a opção de cabeçalho e itens.

**Sintaxe: AxCadastro(cAlias, cTitulo, cVldExc, cVldAlt)**

**Parâmetros:**

<b>cAlias</b>	Alias padrão do sistema para utilização, o qual deve estar definido no dicionário de dados – SX3.
<b>cTitulo</b>	Título da Janela
<b>cVldExc</b>	Validação para Exclusão
<b>cVldAlt</b>	Validação para Alteração

**Exemplo: Função AxCadastro()**

```
#include "protheus.ch"
```

```
/*
```

```

+-----
|-----|
| Função      | XCADSA2      | Autor | ARNALDO RAYMUNDO JR. | Data |
|-----|
|-----|
| Descrição          | Exemplo de utilização da função AXCADASTRO()
|-----|
| Uso            | Curso ADVPL
|-----|
|-----|
/*/

User Function XCadSA2()

Local cAlias      := "SA2"
Local cTitulo     := "Cadastro de Fornecedores"
Local cVldExc    := ".T."
Local cVldAlt    := ".T."

dbSelectArea(cAlias)
dbSetOrder(1)
AxCadastro(cAlias,cTitulo,cVldExc,cVldAlt)

Return Nil

```

### Eercício 01

Desenvolver um AxCadastro para a tabela padrão do ERP – SB1: Produtos

#### **Exemplo: Função de validação da alteração**

```

/*
+-----
|-----|
| Função      | VLDALT      | Autor | ARNALDO RAYMUNDO JR. | Data |
|-----|
|-----|
| Descrição          | Função de validação de alteração para a
AXCADASTRO()   |
+-----|
-----|

```

```

| Uso          | Curso ADVPL
|
+-----+
-----
*/
User Function VldAlt(cAlias,nReg,nOpc)

Local lRet      := .T.
Local aArea     := GetArea()
Local nOpcao    := 0

nOpcao := AxAltera(cAlias,nReg,nOpc)

If nOpcao == 1
    MsgInfo("Ateração concluída com sucesso!")
Endif

RestArea(aArea)

Return lRet

```

### **Exemplo: Função de validação da exclusão**

---

```

/*
+-----+
-----
| Função      | VLDEXC      | Autor | ARNALDO RAYMUNDO JR. | Data |
|
+-----+
-----
| Descrição    | Função de validação de exclusão para a
AXCADASTRO()   |
|
+-----+
-----
| Uso          | Curso ADVPL
|
+-----+
-----
*/
User Function VldExc(cAlias,nReg,nOpc)

Local lRet      := .T.
Local aArea     := GetArea()
Local nOpcao    := 0

nOpcao := AxExclui(cAlias,nReg,nOpc)

```

```
If nOpcão == 1
    MsgInfo("Exclusão concluída com sucesso!")
Endif

RestArea(aArea)
Return lRet
```



### Exercícios

#### Exercício 02

Implementar no AxCadastro as validações de alteração e exclusão.

## 2.9 Mbrowse()

A Mbrowse() é uma funcionalidade de cadastro que permite a utilização de recursos mais aprimorados na visualização e manipulação das informações do sistema, possuindo os seguintes componentes:

- ② Browse padrão para visualização das informações da base de dados, de acordo com as configurações do SX3 – Dicionário de Dados (campo browse).
- ② Parametrização para funções específicas para as ações de visualização, inclusão, alteração e exclusão de informações, o que viabiliza a manutenção de informações com estrutura de cabeçalhos e itens.
- ② Recursos adicionais como identificadores de status de registros, legendas e filtros para as informações.

**Sintaxe: MBrowse(nLin1, nCol1, nLin2, nCol2, cAlias, aFixe, cCpo, nPar08, cFun, nClickDef, aColors, cTopFun, cBotFun, nPar14, bInitBloc, lNoMnuFilter, lSeeAll, lChgAll)**

**Parâmetros:**

<b>nLin1</b>	Número da Linha Inicial
<b>nCol1</b>	Número da Coluna Inicial
<b>nLin2</b>	Número da Linha Final
<b>nCol2</b>	Número da Coluna Final
<b>cAlias</b>	<p>Alias do arquivo que será visualizado no browse. Para utilizar a função MBrowse com arquivos de trabalho, o alias do arquivo de trabalho deve ser obrigatoriamente 'TRB' e o parâmetro aFixe torna-se obrigatório.</p>
<b>aFixe</b>	<p>Array bi-dimensional contendo os nomes dos campos fixos pré-definidos, obrigando a exibição de uma ou mais colunas ou a definição das colunas quando a função é utilizada com arquivos de trabalho. A estrutura do array é diferente para arquivos que fazem parte do dicionário de dados e para arquivos de trabalho.</p> <p>Arquivos que fazem parte do dicionários de dados</p> <p>[n][1]=&gt;Descrição do campo [n][2]=&gt;Nome do campo</p> <p>Arquivos de trabalho</p> <p>[n][1]=&gt;Descrição do campo [n][2]=&gt;Nome do campo [n][3]=&gt;Tipo [n][4]=&gt;Tamanho [n][5]=&gt;Decimal [n][6]=&gt;Picture</p>

**Parâmetros:**

<b>cCpo</b>	Campo a ser validado se está vazio ou não para exibição do bitmap de status. Quando esse parâmetro é utilizado, a primeira coluna do browse será um bitmap indicando o status do registro, conforme as condições configuradas nos parâmetros <b>cCpo</b> , <b>cFun</b> e <b>aColors</b> .
<b>nPar08</b>	Parâmetro reservado.
<b>cFun</b>	Função que retornará um valor lógico para exibição do bitmap de status. Quando esse parâmetro é utilizado, o parâmetro <b>cCpo</b> é automaticamente desconsiderado.
<b>nClickDef</b>	Número da opção do aRotina que será executada quando for efetuado um duplo clique em um registro do browse. O default é executar a rotina de visualização.
<b>aColors</b>	Array bi-dimensional para possibilitar o uso de diferentes bitmaps de status. [n][1]=>Função que retornará um valor lógico para a exibição do bitmap [n][2]=>Nome do bitmap que será exibido quando a função retornar .T. (True). O nome do bitmap deve ser um recurso do repositório e quando esse parâmetro é utilizado os parâmetros <b>cCpo</b> e <b>cFun</b> são automaticamente desconsiderados.
<b>cTopFun</b>	Função que retorna o limite superior do filtro baseado na chave de índice selecionada. Esse parâmetro deve ser utilizado em conjunto com o parâmetro <b>cBotFun</b> .
<b>cBotFun</b>	Função que retorna o limite inferior do filtro baseado na chave de índice selecionada. Esse parâmetro deve ser utilizado em conjunto com o parâmetro <b>cTopFun</b> .
<b>nPar14</b>	Parâmetro reservado.
<b>bInitBloc</b>	Bloco de código que será executado no ON INIT da janela do browse. O bloco de código receberá como parâmetro o objeto da janela do browse.
<b>lNoMnuFilter</b>	Valor lógico que define se a opção de filtro será exibida no menu da MBrowse. .T. => Não exibe a opção no menu ou .F. => (default) Exibe a opção no menu. A opção de filtro na MBrowse está disponível apenas para TOPConnect.
<b>ISeeAll</b>	Identifica se o Browse deverá mostrar todas as filiais. O valor default é .F. ( False ), não mostra todas as filiais. Caso os parâmetros <b>cTopFun</b> ou <b>cBotFun</b> sejam informados esse parâmetro será configurado automaticamente para .F. ( False ). Parâmetro válido à partir da <b>versão 8.11</b> . A função <b>SetBrwSeeAll</b> muda o valor default desse parâmetro.
<b>lChgAll</b>	Identifica se o registro de outra filial está autorizado para alterações. O valor default é .F. ( False ), não permite alterar registros de outras filiais. Quando esse parâmetro está configurado para .T. ( True ), o parâmetro <b>ISeeAll</b> é configurado automaticamente para .T. ( True ). Caso os parâmetros <b>cTopFun</b> ou <b>cBotFun</b> sejam informados esse parâmetro será configurado automaticamente para .F. ( False ). Parâmetro válido à partir da <b>versão 8.11</b> . A função <b>SetBrwChgAll</b> muda o valor default desse parâmetro.

## Variáveis private adicionais

	Array contendo as funções que serão executadas pela Mbrowse, nele será definido o tipo de operação a ser executada (inclusão, alteração, exclusão, visualização, pesquisa, etc. ), e sua estrutura é composta de 5 (cinco) dimensões:  [n][1] - Título; [n][2] – Rotina; [n][3] – Reservado; [n][4] – Operação (1 - pesquisa; 2 - visualização; 3 - inclusão; 4 - alteração; 5 - exclusão); Ele ainda pode ser parametrizado com as funções básicas da AxCadastro conforme abaixo:  AADD(aRotina,{"Pesquisar" , "AxPesqui",0,1}) AADD(aRotina,{"Visualizar" , "AxVisual",0,2}) AADD(aRotina,{"Incluir" , "AxInclui",0,3}) AADD(aRotina,{"Alterar" , "AxAltera",0,4}) AADD(aRotina,{"Excluir" , "AxDeleta",0,5})
cCadastro	Título do browse que será exibido.

### Informações passadas para funções do aRotina:

Ao definir as funções no array aRotina, se o nome da função não for especificado com "()", a Mbrowse passará como parâmetros as seguintes variáveis de controle:

cAlias	Nome da área de trabalho definida para a Mbrowse
nReg	Recno do registro posicionado no Browse
nOpc	Posição da opção utilizada na Mbrowse de acordo com a ordem da função no array a Rotina.

A posição das funções no array aRotina define o conteúdo de uma variável de controle que será repassada para as funções chamadas a partir da Mbrowse, convencionada como nOpc. Desta forma, para manter o padrão da aplicação ERP a ordem a ser seguida na definição do aRotina é:



*Importante*

1. Pesquisar
2. Visualizar
3. Incluir
4. Alterar
5. Excluir
6. Livre

### **Exemplo: Função Mbrowse()**

---

```
#include "protheus.ch"

/*
+-----
| Função      | MBRWSA1      | Autor | ARNALDO RAYMUNDO JR. | Data |
|
+-----
| Descrição          | Exemplo de utilização da função MBROWSE()
|
+-----
| Uso              | Curso ADVPL
|
+-----
/*

User Function MBrwSA1()

Local cAlias           := "SA1"
Private cCadastro       := "Cadastro de Clientes"
Private aRotina := {}

AADD(aRotina,{"Pesquisar"}           , "AxPesqui",0,1})
AADD(aRotina,{"Visualizar"}          , "AxVisual",0,2})
AADD(aRotina,{"Incluir"}             , "AxInclui",0,3})
AADD(aRotina,{"Alterar"}             , "AxAltera",0,4})
AADD(aRotina,{"Excluir"}             , "AxDelete",0,5})

dbSelectArea(cAlias)
dbSetOrder(1)
mBrowse(6,1,22,75,cAlias)

Return Nil
```

---

### **Exemplo: Função Inclui() substituindo a função AxInclui() – Chamada da Mbrowse()**

---

```
#include "protheus.ch"

/*
```

```

+-----
| Função      | MBRWSA1      | Autor | ARNALDO RAYMUNDO JR. | Data |
|             |
+-----
```

```

| Descrição      | Exemplo de utilização da função MBROWSE()
|             |
+-----
```

```

| Uso          | Curso ADVPL
|             |
+-----
```

```

*/

```

#### User Function MBrwSA1()

```

Local cAlias           := "SA1"
Private cCadastro      := "Cadastro de Clientes"
Private aRotina := {}

AADD(aRotina,{"Pesquisar"}      ,,"AxPesqui" ,0,1})
AADD(aRotina,{"Visualizar"}     ,,"AxVisual" ,0,2})
AADD(aRotina,{"Incluir"}        ,,"U_Inclui"   ,0,3})
```

#### Exemplo (continuação):

```

AADD(aRotina,{"Alterar"}      ,,"AxAltera" ,0,4})
AADD(aRotina,{"Excluir"}       ,,"AxDelete" ,0,5})

dbSelectArea(cAlias)
dbSetOrder(1)
mBrowse(6,1,22,75,cAlias)

Return Nil

```

#### Exemplo: Função Inclui() substituindo a função AxInclui() – Função Inclui()

```

/*
+-----
```

```

| Função      | INCLUI      | Autor | ARNALDO RAYMUNDO JR. | Data |
|             |
+-----
```

```

| Descrição      | Função de inclusão específica chamando a
AXINCLUI()|

```

```

+-----
| Uso          | Curso ADVPL
|
+-----
|-----|
| */          |
|-----|
User Function Inclui(cAlias, nReg, nOpc)

Local cTudoOk := "(Alert('OK'),.T.)"
Local nOpc := 0

nOpc := AxInclui(cAlias,nReg,nOpc,,,cTudoOk)

If nOpc == 1
    MsgBox("Inclusão concluída com sucesso!")
Elseif      == 2
    MsgBox("Inclusão cancelada!")
Endif

Return Nil

```

#### **Exemplo: Determinando a opção do aRotina pela informação recebida em nOpc**

```

#include "protheus.ch"

/*
+-----
| Função      | EXCLUI        | Autor | ARNALDO RAYMUNDO JR. | Data |
|
+-----
| Descrição    | Função de exclusão específica chamando a AxDelete
|
+-----
| Uso          | Curso ADVPL
|
+-----
| */          |

User Function Exclui(cAlias, nReg, nOpc)

Local cTudoOk := "(Alert('OK'),.T.)"
Local nOpc := 0

nOpc := AxDelete(cAlias,nReg,aRotina[nOpc,4])

```

```

// Identifica corretamente a opção definida para o função em
aRotinas com mais // do que os 5 elementos padrões.

If nOpc == 1
    MsgBox("Exclusão realizada com sucesso!")
Elseif      == 2
    MsgBox("Exclusão cancelada!")
Endif

Return Nil

```

### 2.9.1 AxFunctions()

Conforme mencionado nos tópicos sobre as interfaces padrões AxCadastro() e Mbrowse(), existem funções padrões da aplicação ERP que permitem a visualização, inclusão, alteração e exclusão de dados em formato simples.

Estas funções são padrões na definição da interface AxCadastro() e podem ser utilizadas também da construção no array aRotina utilizado pela Mbrowse(), as quais estão listadas a seguir:

[? AXPESQUI\(\)](#)

[? AXVISUAL\(\)](#)

[? AXINCLUI\(\)](#)

[? AXALTERA\(\)](#)

[? AXDELETA\(\)](#)

**AXPESQUI()**

Sintaxe	AXPESQUI()
Descrição	Função de pesquisa padrão em registros exibidos pelos browses do sistema, a qual posiciona o browse no registro pesquisado. Exibe uma tela que permite a seleção do índice a ser utilizado na pesquisa e a digitação das informações que compõe a chave de busca.

**AXVISUAL()**

Sintaxe	<b>AXVISUAL(cAlias, nReg, nOpc, aAcho, nColMens, cMensagem, cFunc,; aButtons, lMaximized )</b>
Descrição	Função de visualização padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

**AXINCLUI()**

Sintaxe	<b>AxInclui(cAlias, nReg, nOpc, aAcho, cFunc, aCpos, cTudoOk, lf3,; cTransact, aButtons, aParam, aAuto, lVirtual, lMaximized)</b>
---------	---

<b>Descrição</b>	Função de inclusão padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().
------------------	---

#### **AXALTERA()**

---

<b>Sintaxe</b>	<b>AxAltera(cAlias, nReg, nOpc, aAcho, cFunc, aCpos, cTudoOk, lf3,; cTransact, aButtons, aParam, aAuto, lVirtual, lMaximized)</b>
<b>Descrição</b>	Função de alteração padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

#### **AXDELETA()**

---

<b>Sintaxe</b>	<b>AXDELETA(cAlias, nReg, nOpc, cTransact, aCpos, aButtons, aParam,; aAuto, lMaximized)</b>
<b>Descrição</b>	Função de exclusão padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().



## Exercícios

### Exercício 03

Implementar uma MBrowse com as funções de cadastro padrões para a tabela padrão do ERP – SB1: Produtos

## 2.9.2 FilBrowse()

A FilBrowse() é uma funcionalidade que permite a utilização de filtros na MBrowse().

**Sintaxe: FilBrowse(cAlias, aQuery, cFiltro, IShowProc)**

### Parâmetros:

<b>cAlias</b>	Alias ativo definido para a Mbrowse() Este parâmetro deverá ser inicializado sempre vazio e sua passagem obrigatoriamente por referência, pois, seu retorno será enviado para a função EndFilBrw().
<b>aQuery</b>	[1]=>Nome do Arquivo Físico [2]=>Ordem correspondente ao Sindex
<b>cFiltro</b>	Condição de filtro para a MBrowse()
<b>IShowProc</b>	Habilita (.T.) ou desabilita (.F.) a apresentação da mensagem “Selecionando registros ...”, no processamento.

## 2.9.3 EndFilBrw()

A EndFilBrw() é uma funcionalidade que permite eliminar o filtro e o arquivo temporário criados pela FilBrowse().

**Sintaxe: EndFilBrw(cAlias, aQuery)**

### Parâmetros:

<b>cAlias</b>	Alias ativo definido para a Mbrowse()
<b>aQuery</b>	Array de retorno passado por referência para a FilBrowse(). [1]=>Nome do Arquivo Físico [2]=>Ordem correspondente ao Sindex

## 2.9.4 PesqBrw()

A PesqBrw() é uma funcionalidade que permite a pesquisa dentro da MBrowse(). Esta função deverá obrigatoriamente substituir a função AxPesqui, no array do aRotina, sempre que for utilizada a função FilBrowse().

**Sintaxe:** PesqBrw(cAlias , nReg, bBrwFilter)

**Parâmetros:**

cAlias	Alias ativo definido para a Mbrowse()
nReg	Número do registro
bBrwFilter	Bloco de Código que contém a FilBrowse() Ex: bBrwFilter := {    FilBrowse(cAlias, aQuery, cFiltro, lShowProc) }

## 2.9.5 BrwLegenda ()

A BrwLegenda() é uma funcionalidade que permite a inclusão de legendas na MBrowse().

**Sintaxe:** BrwLegenda(cCadastro , cTitulo, aLegenda)

**Parâmetros:**

cCadastro	Mesma variável utilizada para a MBrowse, que identifica o cadastro que está em uso no momento
cTitulo	Título (identificação) da Legenda
aLegenda	Array contendo de definição da cor e do texto, explicativo sobre o que ela representa na MBrowse Ex: [{"Cor": "Texto"}]

### **Lista de cores disponíveis no Protheus**



*Importante*

- ☒ BR\_AMARELO
- ☒ BR\_AZUL
- ☒ BR\_BRANCO
- ☒ BR\_CINZA
- ☒ BR\_LARANJA
- ☒ BR\_MARRON
- ☒ BR\_VERDE
- ☒ BR\_VERMELHO
- ☒ BR\_PINK
- ☒ BR\_PRETO

## **Exemplo: Mbrowse() utilizando as funções acessórias**

---

```
#Include "Protheus.ch"

/*
+-----
| Programa | MBrwSA2    | Autor | SERGIO FUZINAKA | Data |
|          |           |
+-----+
| Descrição      | Exemplo da MBrowse utilizando a tabela de
Cadastro de |
|          | Fornecedores
|          |
+-----+
| Uso          | Curso de ADVPL
|          |
+-----+
*/
User Function MBrwSA2()

Local cAlias := "SA2"
Local aCores := {}
Local cFiltrar := "A2_FILIAL == '"+xFilial('SA2')+"' .And. A2_EST ==
'SP'"

Private cCadastro := "Cadastro de Fornecedores"
Private aRotina := {}
Private aIndexSA2 := {}
Private bFiltrarBrw:= { || FilBrowse(cAlias,@aIndexSA2,@cFiltrar) }

AADD(aRotina,{"Pesquisar" , "PesqBrw" ,0,1})
AADD(aRotina,{"Visualizar" , "AxVisual" ,0,2})
AADD(aRotina,{"Incluir" , "U_BInclui" ,0,3})
AADD(aRotina,{"Alterar" , "U_BAltera" ,0,4})
AADD(aRotina,{"Excluir" , "U_BDelete" ,0,5})
AADD(aRotina,{"Legenda" , "U_BLegenda" ,0,3})

/*
-- CORES DISPONIVEIS PARA LEGENDA --
BR_AMARELO
BR_AZUL
BR_BRANCO
BR_CINZA
```

```

BR_LARANJA
BR_MARRON
BR_VERDE
BR_VERMELHO
BR_PINK
BR_PRETO
*/
AADD(aCores,{"A2_TIPO == 'F'" , "BR_VERDE" })
AADD(aCores,{"A2_TIPO == 'J'" , "BR_AMARELO" })
AADD(aCores,{"A2_TIPO == 'X'" , "BR_LARANJA" })
AADD(aCores,{"A2_TIPO == 'R'" , "BR_MARRON" })
AADD(aCores,{"Empty(A2_TIPO)" , "BR_PRETO" })

dbSelectArea(cAlias)
dbSetOrder(1)

//+-----
//| Cria o filtro na MBrowse utilizando a função FilBrowse
//+-----
Eval(bFiltrarBrw)

dbSelectArea(cAlias)
dbGoTop()
mBrowse(6,1,22,75,cAlias,,,,,aCores)

//+-----
//| Deleta o filtro utilizado na função FilBrowse
//+-----
EndFilBrw(cAlias,aIndexSA2)

Return Nil

//+-----
//| Função: BInclui - Rotina de Inclusão
//+-----
User Function BInclui(cAlias,nReg,nOpc)

Local nOpcao := 0

nOpcao := AxInclui(cAlias,nReg,nOpc)

If nOpcao == 1
    MsgInfo("Inclusão efetuada com sucesso!")
Else
    MsgInfo("Inclusão cancelada!")
Endif

Return Nil

```

```

//+-----
//|Função: BAltera - Rotina de Alteração
//+-----
User Function BAltera(cAlias,nReg,nOpc)

Local nOpcao := 0

nOpcao := AxAltera(cAlias,nReg,nOpc)

If nOpcao == 1
    MsgInfo("Alteração efetuada com sucesso!")
Else
    MsgInfo("Alteração cancelada!")
Endif

Return Nil
//+-----
//|Função: BDeleta - Rotina de Exclusão
//+-----
User Function BDeleta(cAlias,nReg,nOpc)

Local nOpcao := 0

nOpcao := AxDeleta(cAlias,nReg,nOpc)

If nOpcao == 1
    MsgInfo("Exclusão efetuada com sucesso!")
Else
    MsgInfo("Exclusão cancelada!")
Endif

Return Nil
//+-----
//|Função: BLegenda - Rotina de Legenda
//+-----
User Function BLegenda()

Local aLegenda := {}

AADD(aLegenda,{"BR_VERDE" , "Pessoa Física" })
AADD(aLegenda,{"BR_AMARELO" , "Pessoa Jurídica" })
AADD(aLegenda,{"BR_LARANJA" , "Exportação" })
AADD(aLegenda,{"BR_MARRON" , "Fornecedor Rural" })
AADD(aLegenda,{"BR_PRETO" , "Não Classificado" })

BrwLegenda(cCadastro, "Legenda", aLegenda)

```

Return Nil



### Exercícios

#### Exercício 04

Implementar a legenda para a MBrowse da tabela padrão do ERP – SB1: Produtos

## 2.10 MarkBrowse()

A função MarkBrow() permite que os elementos de um browse, sejam marcados ou desmarcados. Para utilização da MarkBrowse() é necessário declarar as variáveis cCadastro e aRotina como Private, antes da chamada da função.

**Sintaxe:** **MarkBrow (cAlias, cCampo, cCpo, aCampos, lInvert, cMarca, cCtrlM, uPar8, cExplIni, cExpFim, cAval, bParBloco)**

#### Parâmetros:

<b>cAlias</b>	Alias ativo definido para a Mbrowse()
<b>cCampo</b>	Campo do arquivo onde será feito o controle (gravação) da marca.
<b>cCpo</b>	Campo onde será feita a validação para marcação e exibição do bitmap de status.
<b>aCampos</b>	Vetor de colunas a serem exibidas no browse, deve conter as seguintes dimensões: [n][1] – nome do campo; [n][2] - Nulo (Nil); [n][3] - Título do campo; [n][4] - Máscara (picture).
<b>lInvert</b>	Inverte a marcação.
<b>cMarca</b>	String a ser gravada no campo especificado para marcação.
<b>cCtrlM</b>	Função a ser executada caso deseje marcar todos elementos.
<b>uPar8</b>	Parâmetro reservado.
<b>cExplIni</b>	Função que retorna o conteúdo inicial do filtro baseada na chave de índice selecionada.
<b>cExpFim</b>	Função que retorna o conteúdo final do filtro baseada na chave de índice selecionada.
<b>cAval</b>	Função a ser executada no duplo clique em um elemento no browse.
<b>bParBloco</b>	Bloco de código a ser executado na inicialização da janela

#### **Informações passadas para funções do aRotina:**

Ao definir as funções no array aRotina, se o nome da função não for especificado com “()”, a MarkBrowse passará como parâmetros as seguintes variáveis de controle:

<b>cAlias</b>	Nome da área de trabalho definida para a Mbrowse
<b>nReg</b>	Recno do registro posicionado no Browse
<b>nOpc</b>	Posição da opção utilizada na Mbrowse de acordo com a ordem da função no array a Rotina.
<b>cMarca</b>	Marca em uso pela MarkBrw()
<b>lInverte</b>	Indica se foi utilizada a inversão da seleção dos itens no browse.

### **2.10.1 Funções de Apoio**

**GetMark:** define a marca atual.

**IsMark:** avalia se um determinado conteúdo é igual a marca atual.

**ThisMark:** captura a marca em uso.

**ThisInv:** indica se foi usado o recurso de selecionar todos (inversão).

**MarkBRefresh:** atualiza exibição na marca do browse. Utilizada quando a marca é colocada ou removida em blocos e não pelo clique.

---

#### **Exemplo: Função MarkBrow() e acessórios**

```
#include "protheus.ch"
/*
+-----
| Programa | MkBrwSA1 | Autor | ARNALDO RAYMUNDO JR. | Data |
|
+-----
| Desc.      | MarkBrowse Genérico
|
+-----
| Uso          | Curso de ADVPL
|
+-----
*/
USER FUNCTION MkBrwSA1()
```

```

Local aCpos           := {}
Local aCampos         := {}
Local nI              := 0
Local cAlias          := "SA1"

Private aRotina        := {}
Private cCadastro      := "Cadastro de Clientes"
Private aRecSel         := {}

AADD(aRotina,{"Pesquisar"} ,,"AxPesqui" ,0,1})
AADD(aRotina,{"Visualizar"} ,,"AxVisual" ,0,2})
AADD(aRotina,{"Incluir"} ,,"AxInclui" ,0,3})
AADD(aRotina,{"Alterar"} ,,"AxAltera" ,0,4})
AADD(aRotina,{"Excluir"} ,,"AxDelete" ,0,5})
AADD(aRotina,{"Visualizar Lote"} ,,"U_VisLote" ,0,5})

AADD(aCpos,      "A1_OK"      )
AADD(aCpos,      "A1_FILIAL" )
AADD(aCpos,      "A1_COD"     )
AADD(aCpos,      "A1_LOJA"    )
AADD(aCpos,      "A1_NOME"   )
AADD(aCpos,      "A1_TIPO"   )

dbSelectArea("SX3")
dbSetOrder(2)
For nI := 1 To Len(aCpos)
  IF dbSeek(aCpos[nI])
    AADD(aCampos,{X3_CAMPO,"",IIF(nI==1,"",Trim(X3_TITULO)),;
                  Trim(X3_PICTURE)})}
  ENDIF
Next

DbSelectArea(cAlias)
DbSetOrder(1)

MarkBrow(cAlias,aCpos[1],"A1_TIPO == '"
" ,aCampos,.F.,GetMark("SA1","A1_OK"))

Return Nil

```

#### **Exemplo: Função VisLote() – utilização das funções acessórias da MarkBrow()**

```

/*
+-----
| Programa | VisLote()           | Autor | ARNALDO RAYMUNDO JR. | Data |
|          |                               |       |                         |       |

```

```

+-----+
| Desc.      | Função utilizada para demonstrar o uso do recurso da
| MarkBrowse |
+-----+
| Uso          | Curso de ADVPL
|
+-----+
|/*|
USER FUNCTION VisLote()
Local cMarca      := ThisMark()
Local nX           := 0
Local lInvert      := ThisInv()
Local cTexto        := ""
Local cEOL := CHR(10)+CHR(13)
Local oDlg
Local oMemo

DbSelectArea("SA1")
DbGoTop()
While SA1->(!EOF())

    // IsMark("A1_OK", cMarca, lInverte)
    IF SA1->A1_OK == cMarca .AND. !lInvert
        AADD(aRecSel,{SA1->(Recno()),SA1->A1_COD, SA1->A1_LOJA, SA1-
>A1_NREDUZ})
    ELSEIF SA1->A1_OK != cMarca .AND. lInvert
        AADD(aRecSel,{SA1->(Recno()),SA1->A1_COD,SA1->A1_LOJA, SA1-
>A1_NREDUZ})
    ENDIF

    SA1->(dbSkip())
Enddo

IF Len(aRecSel) > 0
    cTexto := "Código | Loja | Nome Reduzido      "+cEol
    //          "1234567890123456789012345678901234567890
    //          "CCCCCC | LL | NNNNNNNNNNNNNNNNNNNNNNNNNN +cEol

    For nX := 1 to Len(aRecSel)

        cTexto += aRecSel[nX][2]+Space(1)+ "|" +Space(2) +
aRecSel[nX][3]+Space(3)+"|"
        cTexto += Space(1)+SUBSTRING(aRecSel[nX][4],1,20)+Space(1)
        cTexto += cEOL

    Next nX

```

```
DEFINE MSDIALOG oDlg TITLE "Clientes Seleccionados" From 000,000
TO 350,400 PIXEL
@ 005,005 GET oMemo  VAR cTexto MEMO SIZE 150,150 OF oDlg PIXEL
oMemo:bRClicked := {| |AllwaysTrue()}
DEFINE SBUTTON  FROM 005,165 TYPE 1 ACTION oDlg:End() ENABLE OF
oDlg PIXEL
ACTIVATE MSDIALOG oDlg CENTER
LimpiaMarca()
ENDIF

RETURN
```

## **Exemplo: Função LimpaMarca() – utilização das funções acessórias da MarkBrowse()**

```
/*
+-----
| Programa | LimpaMarca | Autor | ARNALDO RAYMUNDO JR. | Data |
|
+-----
| Desc.      | Função utilizada para demonstrar o uso do recurso da
MarkBrowse|
+
| Uso          | Curso de ADVPL
|
+-----
*/
STATIC FUNCTION LimpaMarca()

Local nX := 0

    For nX := 1 to Len(aRecSel)
        SA1->(DbGoto(aRecSel[nX][1]))
        RecLock("SA1",.F.)
        SA1->A1_OK := SPACE(2)
        MsUnLock()
    Next nX
RETURN
```



### **Exercício 05**

Implementar uma MarkBrowse com as funções de cadastro padrões para a tabela padrão do ERP – SB1: Produtos

### **Exercício 06**

Implementar na MarkBrowse da tabela padrão SB1, uma função para exclusão de múltiplos itens selecionados no Browse. ERP – SB1: Produtos

### **Exercício 07**

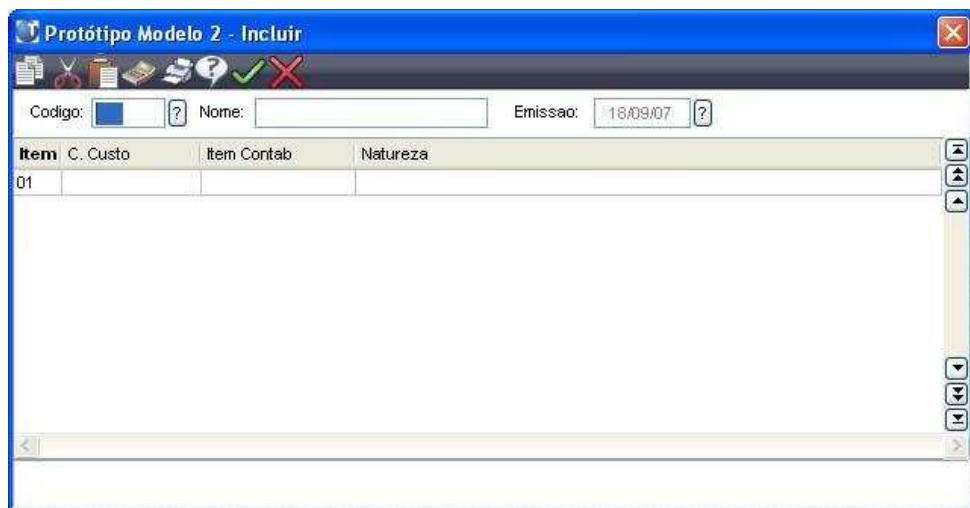
Implementar uma MarkBrowse para a tabela SA1, para visualização de dados de

múltiplos clientes selecionados

## 2.11 Modelo2()

O nome Modelo 2 foi conceituado pela Microsiga por se tratar de um protótipo de tela para entrada de dados. Inicialmente vamos desmistificar dois pontos:

- ☒ **Função Modelo2()** – Trata-se de uma função pronta que contempla o protótipo Modelo 2, porém, este é um assunto que não iremos tratar aqui, visto que é uma funcionalidade simples que quando necessário intervir em algo na rotina não há muito recurso para tal.
- ☒ **Protótipo Modelo 2** – Trata-se de uma tela, como a figura abaixo, onde seu objetivo é efetuar a manutenção em vários registros de uma só vez. Por exemplo: efetuar o movimento interno de vários produtos do estoque em um único lote.

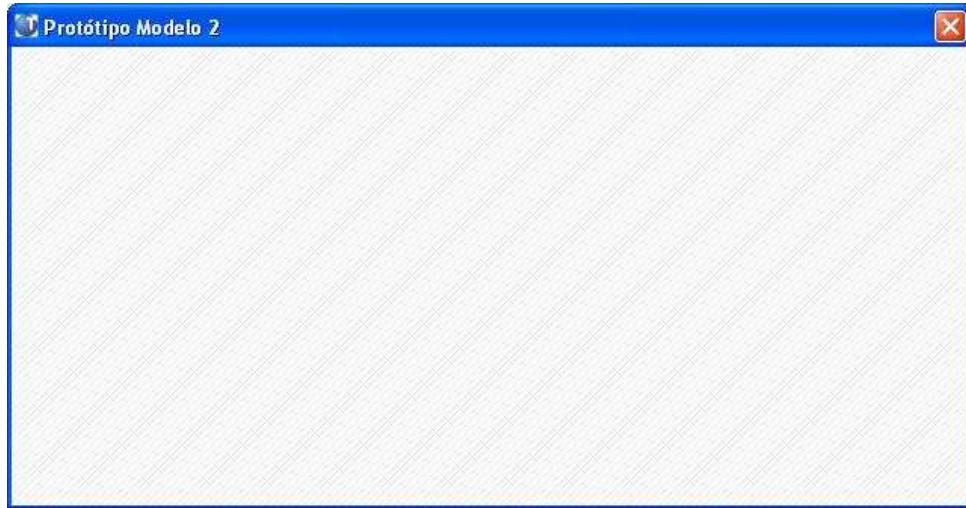


### 2.11.1 Componentes de uma tela no formato Modelo 2

## Objeto MsDialog()

---

Deve ser utilizada como janela padrão para entrada de dados, é um tipo de objeto modal, ou seja, não permite que outra janela ativa receba dados enquanto esta estiver ativa.



Toda vez que utilizar este comando, o ADVPL exige que seja declarada a diretiva "Include" no cabeçalho do programa o arquivo "Protheus.ch", isto porque o compilador precisará porque este comando trata-se de um pseudo código e sua tradução será feita na compilação. Vale lembrar, também, que este só será acionado depois que instanciado e ativado por outro comando.

```
DEFINE MSDIALOG oDlg TITLE "Protótipo Modelo 2" FROM 0,0 TO 280,552
OF;
oMainWnd PIXEL

ACTIVATE MSDIALOG oDlg CENTER
```

Reparem que o comando DEFINE MSDIALOG instanciou e o comando ACTIVATE MSDIALOG ativa todos os objetos, ou seja, todo ou qualquer outro objeto que precisar colocar nesta janela será preciso informar em qual objeto. Para este caso, sempre será utilizada a variável de objeto exportável **oDlg**.

## Função EnchoiceBar()

---

Função que cria uma barra de botões padrão de Ok e Cancelar, permitindo a implementação de botões adicionais.



Sintaxe: ENCHOICEBAR( *oDlg*, *bOk*, *bCancelar*, [ *IMensApag* ], [ *aBotoes* ] )

### Parâmetros:

<b><i>oDlg</i></b>	Objeto	Janela onde a barra será criada.
<b><i>bOk</i></b>	Objeto	Bloco de código executado quando clicado botão Ok.
<b><i>bCancelar</i></b>	Objeto	Bloco de código executado quando clicado.
<b><i>IMensApag</i></b>	Lógico	Indica se ao clicar no botão Ok aparecerá uma tela de <u>confirmação de exclusão</u> . Valor padrão falso.
<b><i>aBotões</i></b>	Vetor	Vetor com informações para criação de botões adicionais na barra. Seu formato é {bitmap, bloco de código, mensagem}.

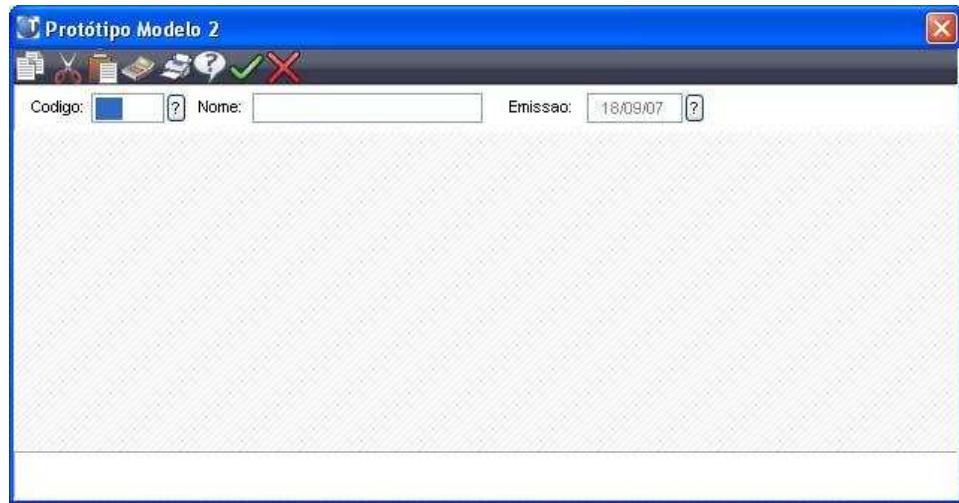


Figura: Protótipo Modelo2 – Enchoice

## Objeto TPanel()

---

Repare que para facilitar o desenvolvimento foi utilizado o objeto TPanel para ajudar o alinhamento dos objetos TSay e TGet, ou seja, a utilização deste recurso permite que o programador não se preocupe com coordenadas complexas para deixar a união dos objetos simétricos.

Utilize o objeto TPanel quando desejar criar um painel estático, onde podem ser criados outros controles com o objetivo de organizar ou agrupar componentes visuais.

**Sintaxe:** `TPanel():New([anRow], [anCol], [acText], [aoWnd], [aoFont],  
[alCentered], [IPar6], [anClrText], [anClrBack], [anWidth],  
[anHeight], [alLowered], [alRaised])`

### Parâmetros:

<b>nRow</b>	Numérico vertical em pixel.
<b>nCol</b>	Numérico horizontal em pixel.
<b>cText</b>	Texto a ser exibido ao fundo.
<b>oWnd</b>	Objeto da janela ou controle onde será criado o objeto.
<b>oFont</b>	Características da fonte do texto que aparecerá ao fundo.
<b>lCentered</b>	Exibe o texto do título centralizado.
<b>IPar6</b>	Reservado.
<b>nClrText</b>	Cor do texto de controle.
<b>nClrBack</b>	Cor do fundo de controle.
<b>nWidth</b>	Largura do controle em pixel.
<b>nHeight</b>	Altura do controle em pixel.
<b>lLowered</b>	Exibe o painel rebaixado em relação ao controle de fundo.
<b>lRaised</b>	Exibe a borda do controle rebaixado em relação ao controle de fundo.

## Comando SAY - Objeto: TSay()

---

O comando SAY ou objeto TSay exibe o conteúdo de texto estático sobre uma janela.

### Sintaxe SAY:

```
@ 4,6 SAY "Código:" SIZE 70,7 PIXEL OF oTPanel1
```

**Sintaxe TSay(): TSay():New([anRow], [anCol], [abText], [aoWnd],  
[acPicture], [aoFont], [IPar7], [IPar8], [IPar9], [alPixels],**

[anClrText], [anClrBack], [anWidth], [anHeight], [IPar15], [IPar16],  
 [IPar17], [IPar18], [IPar19])

**Parâmetros:**

<b>anRow</b>	Numérico, opcional. Coordenada vertical em pixels ou caracteres.
<b>anCol</b>	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
<b>abText</b>	Code-Block, opcional. Quando executado deve retornar uma cadeia de caracteres a ser exibida.
<b>aoWnd</b>	Objeto, opcional. Janela ou diálogo onde o controle será criado.
<b>acPicture</b>	Caractere, opcional. Picture de formatação do conteúdo a ser exibido.
<b>aoFont</b>	Objeto, opcional. Objeto tipo tFont para configuração do tipo de fonte que será utilizado para exibir o conteúdo.
<b>IPar7</b>	Reservado.
<b>IPar8</b>	Reservado.
<b>IPar9</b>	Reservado.
<b>alPixels</b>	Lógico, opcional. Se .T. considera coordenadas passadas em pixels se .F., padrão, considera as coordenadas passadas em caracteres.
<b>anClrText</b>	Numérico, opcional. Cor do conteúdo do controle.
<b>anClrBack</b>	Numérico, opcional. Cor do fundo do controle.
<b>anWidth</b>	Numérico, opcional. Largura do controle em pixels.
<b>anHeight</b>	Numérico, opcional. Altura do controle em pixels.
<b>IPar15</b>	Reservado.
<b>IPar16</b>	Reservado.
<b>IPar17</b>	Reservado.
<b>IPar18</b>	Reservado.
<b>IPar19</b>	Reservado.

---

**Comando MSGET - Objeto: TGet()**

---

O comando MsGet ou o objeto TGet é utilizado para criar um controle que armazene ou altere o conteúdo de uma variável através de digitação. O conteúdo da variável só é modificado quando o controle perde o foco de edição para outro controle.

**Sintaxe MSGET:**

**@ 3,192 MSGET dData PICTURE "99/99/99" SIZE 40,7 PIXEL OF oTPanel1**

**Sintaxe TGet():New([anRow], [anCol], [abSetGet], [aoWnd], [anWidth],**

**[anHeight], [acPict], [abValid], [anClrFore], [anClrBack], [aoFont],  
 [IPar12], [oPar13], [alPixel], [cPar15], [IPar16], [abWhen],**

[IPar18], [IPar19], [abChange], [alReadOnly], [alPassword],  
 [cPar23], [acReadVar], [cPar25], [IPar26], [nPar27], [IPar28])

**Parâmetros:**

<b>anRow</b>	Numérico, opcional. Coordenada vertical em pixels ou caracteres.
<b>anCol</b>	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
<b>abSetGet</b>	Bloco de código, opcional. Bloco de código no formato {  u   IF( Pcount( )>0, <var>:= u, <var> ) } que o controle utiliza para atualizar a variável <var>. <var> deve ser tipo caracter, numérico ou data.
<b>aoWnd</b>	Objeto, opcional. Janela ou controle onde o controle será criado.
<b>anWidth</b>	Numérico, opcional. Largura do controle em pixels.
<b>anHeight</b>	Numérico, opcional. Altura do controle em pixels.
<b>acPict</b>	Caractere, opcional. Máscara de formatação do conteúdo a ser exibido.
<b>abValid</b>	Bloco de código, opcional. Executado quando o conteúdo do controle deve ser validado, deve retornar "T" se o conteúdo for válido e "F" quando o conteúdo for inválido.
<b>anClrFore</b>	Numérico, opcional. Cor de fundo do controle.
<b>anClrBack</b>	Numérico, opcional. Cor do texto do controle.
<b>aoFont</b>	Objeto, opcional. Objeto tipo tFont utilizado para definir as características da fonte utilizada, para exibir o conteúdo do controle.
<b>IPar12</b>	Reservado.
<b>oPar13</b>	Reservado.
<b>alPixel</b>	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
<b>cPar15</b>	Reservado.
<b>IPar16</b>	Reservado.
<b>abWhen</b>	Bloco de código, opcional. Executado quando mudança de foco de entrada de dados está sendo efetuada na janela onde o controle foi criado. O bloco deve retornar "T" se o controle deve permanecer habilitado ou "F", se não.
<b>IPar18</b>	Reservado.
<b>IPar19</b>	Reservado.
<b>abChange</b>	Bloco de código, opcional. Executado quando o controle modifica o valor da variável associada.
<b>alReadOnly</b>	Lógico, opcional. Se .T. o controle não poderá ser editado.
<b>alPassword</b>	Lógico, opcional. Se .T. o controle exibirá asteriscos "*" no lugar dos caracteres exibidos pelo controle para simular entrada de senha.
<b>cPar23</b>	Reservado.
<b>acReadVar</b>	Caractere, opcional. Nome da variável que o controle deverá manipular. Deverá ser a mesma variável informada no parâmetro

	abSetGet, e será o retorno da função ReadVar( ).
cPar25	Reservado.
IPar26	Reservado.
nPar27	Reservado.
IPar18	Reservado.

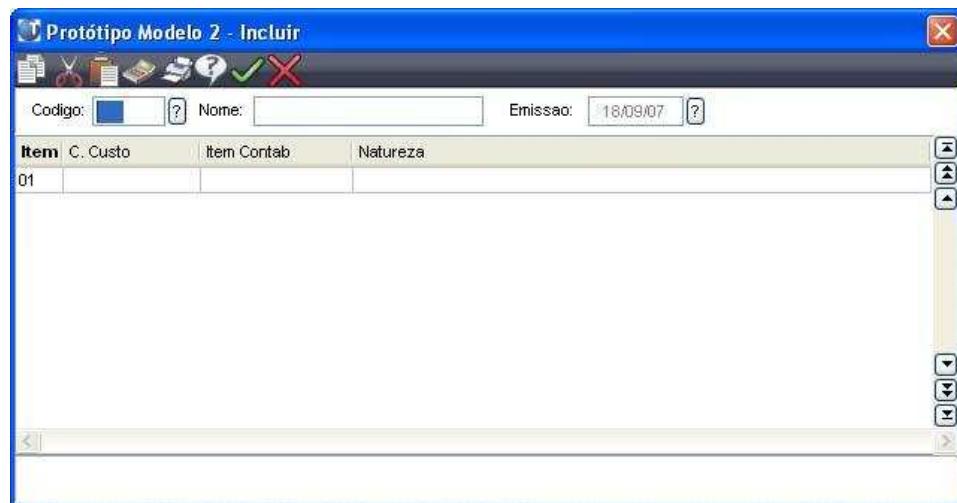
### Objeto MsGetDados()

---

Objeto tipo lista com uma ou mais colunas para cadastramento de dados baseado em um vetor. Sua utilização exige que sejam utilizadas três variáveis com seu escopo “Private”, são elas: aRotina, aHeader e aCOLS.

#### Observações importantes:

- O vetor aHeader deve ser construído com base no dicionário de dados.
- O vetor aCOLS deve ser construído com base no vetor aHeader, porém deve-se criar uma coluna adicional para o controle de exclusão do registro, ou seja, quando o usuário teclar a tecla <DELETE> a linha ficará na cor cinza e esta coluna estará com o seu valor igual a verdadeiro (.T.).
- Quando instanciado este objeto é possível saber em que linha o usuário está porque o objeto trabalha com uma variável de escopo “Public” denominada “n”, seu valor é numérico e terá sempre no conteúdo a linha em que o usuário encontra-se com o cursor.



**Sintaxe: MSGETDADOS():NEW( nSuperior, nEsquerda, nInferior, nDireita,  
nOpc, [ cLinhaOk ], [ cTudoOk ], [ cIniCpos ], [ lApagar ], [ aAlter ],  
, [ uPar1 ], [ lVazio ], [ nMax ], [ cCampoOk ], [ cSuperApagar ],  
[ uPar2 ], [ cApagaOk ], [ oWnd ] )**

**Parâmetros:**

<b>nSuperior</b>	Distância entre a MsGetDados e o extremidade superior do objeto que a contém.
<b>nEsquerda</b>	Distância entre a MsGetDados e o extremidade esquerda do objeto que a contém.
<b>nInferior</b>	Distância entre a MsGetDados e o extremidade inferior do objeto que a contém.
<b>nDireita</b>	Distância entre a MsGetDados e o extremidade direita do objeto que a contém.
<b>nOpc</b>	Posição do elemento do vetor aRotina que a MsGetDados usará como referencia.
<b>cLinhaOk</b>	Função executada para validar o contexto da linha atual do aCols.
<b>cTudoOk</b>	Função executada para validar o contexto geral da MsGetDados (todo aCols).
<b>cIniCpos</b>	Nome dos campos do tipo caracter que utilizarão incremento automático. Este parâmetro deve ser no formato "+<nome do primeiro campo>+<nome do segundo campo>+...".
<b>lApagar</b>	Habilita deletar linhas do aCols. Valor padrão falso.
<b>aAlter</b>	Vetor com os campos que poderão ser alterados.
<b>uPar1</b>	Parâmetro reservado.
<b>lVazio</b>	Habilita validação da primeira coluna do aCols para que esta não esteja vazia. Valor padrão falso.
<b>nMax</b>	Número máximo de linhas permitidas. Valor padrão 99.
<b>cCampoOk</b>	Função executada na validação do campo.
<b>cSuperApagar</b>	Função executada quando pressionada as teclas <Ctrl>+<Delete>.
<b>uPar2</b>	Parâmetro reservado.
<b>cApagaOk</b> <b>oWnd</b>	Função executada para validar a exclusão de uma linha do aCols. Objeto no qual a MsGetDados será criada.

**Variável Private aRotina**

Array com as rotinas que serão executadas na MBrowse e que definirá o tipo de operação que está sendo executada, por exemplo: Pesquisar, Visualizar, Incluir, Alterar, Excluir e outros.

Este vetor precisa ser construído no formato:

<b>Elemento</b>	<b>Conteúdo</b>
1	Título da opção.
2	Nome da rotina (Function).
3	Reservado.
4	Operação (1-Pesquisar;2-Visualizar;3-Incluir;4-Alterar;5-Exclusão).
5	Acesso relacionado a rotina. Se esta opção não for informada, nenhum

	acesso será validado.
--	-----------------------

### **Variável Private aHeader**

---

Array com informações das colunas, ou seja, com as características dos campos que estão contidas no dicionário de dados (SX3), este vetor precisa estar no formato abaixo:

Elemento	Conteúdo
1	Título do campo
2	Nome do campo
3	Máscara do campo
4	Tamanho do campo
5	Decimal do campo
6	Validação de usuário do campo
7	Uso do campo
8	Tipo do campo (caractere, numérico, data e etc.)
9	Prefixo da tabela
10	Contexto do campo (real ou virtual)

### **Variável Private aCols**

---

Vetor com as linhas a serem editadas. As colunas devem ser construídas com base no vetor aHeader e mais uma última coluna com o valor lógico que determina se a linha foi excluída, inicialmente esta deverá ter o seu conteúdo igual a falso ("F").

## **2.11.2 Estrutura de um programa utilizando a Modelo2()**

O exemplo abaixo demonstra a montagem de um programa para a utilização do protótipo Modelo 2. Antes de iniciarmos o exemplo vamos estruturar o programa.

## Estrutura do programa

---

Linhas Programa	
1	<b>Função principal;</b>
2	Declaração e atribuição de variáveis;
3	Acesso a tabela principal e sua ordem;
4	Chamada da função MBrowse;
5	<b>Fim da função principal.</b>
6	
7	<b>Função de visualização, alteração e exclusão;</b>
8	Declaração e atribuição de variáveis;
9	Acesso ao primeiro registro da chave em que está posicionado na MBrowse;
10	Montagem das variáveis estáticas em tela;
11	Montagem do vetor aHeader por meio do dicionário de dados;
12	Montagem do vetor aCOLS de todos os registros referentes à chave principal em que está posicionado na MBrowse;
13	Instância da MsDialog;
14	Instância dos objetos TSay e TGet;
15	Instância do objeto MsGetDados;
16	Ativar o objeto principal que é o objeto da janela;
17	Se for operação diferente de visualização e clicou no botão OK;
18	A operação é de Alteração?
19	Chamar a função para alterar os dados;
20	Caso contrário:
21	Chamar a função para excluir os dados;
22	<b>Fim da função de visualização, alteração e exclusão.</b>
23	
24	<b>Função de inclusão;</b>
25	Declaração e atribuição de variáveis;
26	Montagem das variáveis estáticas em tela;
27	Montagem do vetor aHeader por meio do dicionário de dados;
28	Montagem do vetor aCOLS com o seu conteúdo conforme o inicializador padrão do campo ou vazio, pois trata-se de uma inclusão;
29	Instância da MsDialog;
30	Instância dos objetos TSay e TGet;
31	Instância do objeto MsGetDados;
32	Ativar o objeto principal que é o objeto da janela;
33	Se clicou no botão OK;
34	Chamar a função para incluir os dados;
35	<b>Fim da função de inclusão.</b>

## **Rotina principal**

```
#include "protheus.ch"

//+-----
---+
/// Rotina | xModelo2 | Autor | Robson Luiz (rleg) | Data |
01.01.2007 |
//+-----
---+
/// Descr. | Função exemplo do protótipo Modelo2.
|
//+-----
---+
/// Uso      | Para treinamento e capacitação.
|
//+-----
---+
User Function xModelo2()
    Private cCadastro := "Protótipo Modelo 2"
    Private aRotina := {}

        AADD( aRotina, {"Pesquisar" , "AxPesqui" ,0,1})
        AADD( aRotina, {"Visualizar" , 'U_Mod2Mnt',0,2})
        AADD( aRotina, {"Incluir"     , 'U_Mod2Inc',0,3})
        AADD( aRotina, {"Alterar"    , 'U_Mod2Mnt',0,4})
        AADD( aRotina, {"Excluir"    , 'U_Mod2Mnt',0,5})

        dbSelectArea("ZA3")
        dbSetOrder(1)
        dbGoTop()

        MBrowse(,,, "ZA3")
Return
```

## **Rotina de inclusão**

```
//+-----
---+
/// Rotina | Mod2Inc | Autor | Robson Luiz (rleg) | Data |
01.01.2007 |
//+-----
---+
/// Descr. | Rotina para incluir dados.
|
//+-----
---+
/// Uso      | Para treinamento e capacitação.
|
//+-----
---+
```

```

User Function Mod2Inc( cAlias, nReg, nOpc )
  Local oDlg
  Local oGet
  Local oTPanel1
  Local oTPanel2

  Local cCodigo := ZA3->ZA3_CODIGO
  Local cNome    := ZA3->ZA3_NOME
  Local dData    := dDataBase

  Private aHeader := {}
  Private aCOLS := {}
  Private aREG := {}

  dbSelectArea( cAlias )
  dbSetOrder(1)

  Mod2aHeader( cAlias )
  Mod2aCOLS( cAlias, nReg, nOpc )

  DEFINE MSDIALOG oDlg TITLE cCadastro From 8,0 To 28,80 OF
oMainWnd

  oTPanel1 := TPanel():New(0,0,"",oDlg,NIL,.T.,;
.F.,NIL,NIL,0,16,.T.,.F.)

  oTPanel1:Align := CONTROL_ALIGN_TOP

  @ 4, 006 SAY "Código:"  SIZE 70,7 PIXEL OF oTPanel1
  @ 4, 062 SAY "Nome:"     SIZE 70,7 PIXEL OF oTPanel1
  @ 4, 166 SAY "Emissao:"  SIZE 70,7 PIXEL OF oTPanel1

  @ 3, 026 MSGET cCodigo F3 "SA3" PICTURE "@!" VALID;
Mod2Vend(cCodigo, @cNome);
  SIZE 030,7 PIXEL OF oTPanel1

  @ 3, 080 MSGET cNome When .F. SIZE 78,7 PIXEL OF oTPanel1
  @ 3, 192 MSGET dData PICTURE "99/99/99" SIZE 40,7 PIXEL
OF
  oTPanel1

  oTPanel2 := TPanel():New(0,0,"",oDlg,NIL,.T.,;
.F.,NIL,NIL,0,16,.T.,.F.)
  oTPanel2:Align := CONTROL_ALIGN_BOTTOM

  oGet := MSGetDados():New(0,0,0,nOpc,"U_Mod2LOk()",;
".T.", "+ZA3_ITEM",.T.)
  oGet:oBrowse:Align := CONTROL_ALIGN_ALLCLIENT

```

```
ACTIVATE MSDIALOG oDlg CENTER ON INIT ;
EnchoiceBar(oDlg,{| | IIF(U_Mod2TOK(), Mod2Grvl(),;
( oDlg:End(), NIL ) }},{| | oDlg:End() })
```

Return

## **Rotina de Visualização, Alteração e Exclusão**

```
//+-----  
---+  
//| Rotina | Mod2Mnt | Autor | Robson Luiz (rleg) | Data |  
01.01.2007 |  
//+-----  
---+  
//| Descr. | Rotina para Visualizar, Alterar e Excluir dados.  
|  
//+-----  
---+  
//| Uso     | Para treinamento e capacitação.  
|  
//+-----  
---+  
  
User Function Mod2Mnt( cAlias, nReg, nOpc )  
  
    Local oDlg  
    Local oGet  
    Local oTPanel1  
    Local oTPanel2  
  
    Local cCodigo := Space(Len(Space(ZA3->ZA3_CODIGO)))  
    Local cNome := Space(Len(Space(ZA3->ZA3_NOME)))  
    Local dData := Ctod(Space(8))  
  
    Private aHeader := {}  
    Private aCOLS := {}  
    Private aREG := {}  
  
    dbSelectArea( cAlias )  
    dbGoTo( nReg )  
  
    cCodigo := ZA3->ZA3_CODIGO  
    cNome   := ZA3->ZA3_NOME  
    cData    := ZA3->ZA3_DATA  
  
    Mod2aHeader( cAlias )  
    Mod2aCOLS( cAlias, nReg, nOpc )  
  
    DEFINE MSDIALOG oDlg TITLE cCadastro From 8,0 To 28,80 OF  
oMainWnd  
  
        oTPane1 := TPanel():New(0,0,"",oDlg,NIL,.T.,;  
.F.,NIL,NIL,0,16,.T.,.F.)  
        oTPane1:Align := CONTROL_ALIGN_TOP  
  
        @ 4, 006 SAY "Código:"  SIZE 70,7 PIXEL OF oTPanel1
```

```

        @ 4, 062 SAY "Nome:"      SIZE 70,7 PIXEL OF oTPanel1
        @ 4, 166 SAY "Emissao:" SIZE 70,7 PIXEL OF oTPanel1

        @ 3, 026 MSGET cCodigo When .F. SIZE 30,7 PIXEL OF
oTPanel1
        @ 3, 080 MSGET cNome    When .F. SIZE 78,7 PIXEL OF
oTPanel1
        @ 3, 192 MSGET dData    When .F. SIZE 40,7 PIXEL OF
oTPanel1

        oTPanel2 := TPanel():New(0,0,"",oDlg,NIL,T.,;
.F.,NIL,NIL,0,16,.T.,.F.)
        oTPanel2:Align := CONTROL_ALIGN_BOTTOM

        If nOpc == 4
            oGet := MSGetDados():New(0,0,0,0,nOpc,"U_Mod2LOk()",;
".T." "+ZA3_ITEM",.T.)
            Else
                oGet := MSGetDados():New(0,0,0,0,nOpc)
            Endif
            oGet:oBrowse:Align := CONTROL_ALIGN_ALLCLIENT

        ACTIVATE MSDIALOG oDlg CENTER ON INIT ;
        EnchoiceBar(oDlg,{|| ( IIF( nOpc==4, Mod2GrvA(), ;
IIF( nOpc==5, Mod2GrvE(), oDlg:End() ) ), oDlg:End() ) },;
{|| | oDlg:End() })

Return

```

## Montagem do array aHeader

```
//+-----  
---+  
//| Rotina | Mod2aHeader | Autor | Robson Luiz (rleg)  
| Data | 01.01.2007 |  
//+-----  
---+  
//| Descr. | Rotina para montar o vetor aHeader.  
|  
//+-----  
---+  
//| Uso     | Para treinamento e capacitação.  
|  
//+-----  
---+  
  
Static Function Mod2aHeader( cAlias )  
    Local aArea := GetArea()  
  
        dbSelectArea("SX3")  
        dbSetOrder(1)  
        dbSeek( cAlias )  
        While !EOF() .And. X3_ARQUIVO == cAlias  
            If X3Uso(X3_USADO) .And. cNivel >= X3_NIVEL  
                AADD( aHeader, { Trim( X3Titulo() );  
                    X3_CAMPO,;  
                    X3_PICTURE,;  
                    X3_TAMANHO,;  
                    X3_DECIMAL,;  
                    X3_VALID,;  
                    X3_USADO,;  
                    X3_TIPO,;  
                    X3_ARQUIVO,;  
                    X3_CONTEXT})  
            Endif  
            dbSkip()  
        End  
        RestArea(aArea)  
Return
```

## Montagem do array aCols

```
//+-----  
---+  
//| Rotina | Mod2aCOLS | Autor | Robson Luiz (rleg) | Data |  
01.01.2007 |  
//+-----  
---+  
//| Descr. | Rotina para montar o vetor aCOLS.  
|  
//+-----  
---+  
//| Uso      | Para treinamento e capacitação.  
|  
//+-----  
---+  
Static Function Mod2aCOLS( cAlias, nReg, nOpc )  
    Local aArea := GetArea()  
    Local cChave := ZA3->ZA3_CODIGO  
    Local nI := 0  
  
    If nOpc <> 3  
        dbSelectArea( cAlias )  
        dbSetOrder(1)  
        dbSeek( xFilial( cAlias ) + cChave )  
        While !EOF() .And. ;  
            ZA3->( ZA3_FILIAL + ZA3_CODIGO ) == xFilial( cAlias ) +  
cChave  
                AADD( aREG, ZA3->( RecNo() ) )  
                AADD( aCOLS, Array( Len( aHeader ) + 1 ) )  
                For nI := 1 To Len( aHeader )  
                    If aHeader[nI,10] == "V"  
                        aCOLS[Len(aCOLS),nI] :=  
CriaVar(aHeader[nI,2],.T.)  
                    Else  
                        aCOLS[Len(aCOLS),nI] :=  
FieldGet(FieldPos(aHeader[nI,2]))  
                    Endif  
                    Next nI  
                    aCOLS[Len(aCOLS),Len(aHeader)+1] := .F.  
                    dbSkip()  
                End  
            Else  
                AADD( aCOLS, Array( Len( aHeader ) + 1 ) )  
                For nI := 1 To Len( aHeader )  
                    aCOLS[1, nI] := CriaVar( aHeader[nI, 2], .T. )  
                Next nI  
                aCOLS[1, GdFieldPos("ZA3_ITEM")] := "01"  
                aCOLS[1, Len( aHeader )+1 ] := .F.  
            Endif
```

Restarea( aArea )

Return

## Efetivação da inclusão

```
//+-----  
---+  
//| Rotina | Mod2Grvl | Autor | Robson Luiz (rleg) | Data |  
01.01.2007 |  
//+-----  
---+  
//| Descr. | Rotina para gravar os dados na inclusão.  
|  
//+-----  
---+  
//| Uso     | Para treinamento e capacitação.  
|  
//+-----  
---+  
Static Function Mod2Grvl()  
    Local aArea := GetArea()  
    Local nI := 0  
    Local nX := 0  
  
        dbSelectArea("ZA3")  
        dbSetOrder(1)  
        For nI := 1 To Len( aCOLS )  
            If ! aCOLS[nI,Len(aHeader)+1]  
                RecLock("ZA3",.T.)  
                ZA3->ZA3_FILIAL := xFilial("ZA3")  
                ZA3->ZA3_CODIGO := cCodigo  
                ZA3->ZA3_DATA    := dData  
                For nX := 1 To Len( aHeader )  
                    FieldPut( FieldPos( aHeader[nX, 2] ),  
aCOLS[nI, nX] )  
                    Next nX  
                    MsUnLock()  
                Endif  
            Next nI  
  
            RestArea(aArea)  
        Return
```

## Efetivação da alteração

---

```
//+-----  
---+  
//| Rotina | Mod2GrvA | Autor | Robson Luiz (rleg) | Data |  
01.01.2007 |  
//+-----  
---+  
//| Descr. | Rotina para gravar os dados na alteração.  
|  
//+-----  
---+  
//| Uso     | Para treinamento e capacitação.  
|  
//+-----  
---+  
Static Function Mod2GrvA()  
    Local aArea := GetArea()  
    Local nI := 0  
    Local nX := 0  
  
        dbSelectArea("ZA3")  
        For nI := 1 To Len( aREG )  
            If nI <= Len( aREG )  
                dbGoTo( aREG[nI] )  
                RecLock("ZA3",.F.)  
                If aCOLS[nI, Len(aHeader)+1]  
                    dbDelete()  
                Endif  
            Else  
                RecLock("ZA3",.T.)  
            Endif  
  
            If !aCOLS[nI, Len(aHeader)+1]  
                ZA3->ZA3_FILIAL := xFilial("ZA3")  
                ZA3->ZA3_CODIGO := cCodigo  
                ZA3->ZA3_DATA    := dData  
                For nX := 1 To Len( aHeader )  
                    FieldPut( FieldPos( aHeader[nX, 2] ),  
aCOLS[nI, nX] )  
                Next nX  
            Endif  
            MsUnLock()  
        Next nI  
        RestArea( aArea )  
    Return
```

## Efetivação da exclusão

```
//+-----  
---+  
//| Rotina | Mod2GrvE | Autor | Robson Luiz (rleg) | Data |  
01.01.2007 |  
//+-----  
---+  
//| Descr. | Rotina para excluir os registros.  
|  
//+-----  
---+  
//| Uso     | Para treinamento e capacitação.  
|  
//+-----  
---+  
Static Function Mod2GrvE()  
    Local nl := 0  
  
        dbSelectArea("ZA3")  
        For nl := 1 To Len( aCOLS )  
            dbGoTo(aREG[nl])  
            RecLock("ZA3",.F.)  
            dbDelete()  
            MsUnLock()  
        Next nl  
Return
```

## Função auxiliar: Validação do código do vendedor

```
//+-----  
---+  
//| Rotina | Mod2Vend | Autor | Robson Luiz (rleg) | Data |  
01.01.2007 |  
//+-----  
---+  
//| Descr. | Rotina para validar o código do vendedor.  
|  
//+-----  
---+  
//| Uso     | Para treinamento e capacitação.  
|  
//+-----  
---+  
Static Function Mod2Vend( cCodigo, cNome )  
    If ExistCpo("SA3",cCodigo) .And. ExistChav("ZA3",cCodigo)  
        cNome :=  
        Posicione("SA3",1,xFilial("SA3")+cCodigo,"A3_NOME")  
        Endif  
    Return(!Empty(cNome))
```

## Função auxiliar: Validação do código do centro de custo na mudança de linha

```

//+-----
---+
//| Rotina | Mod2LOk | Autor | Robson Luiz (rleg)    | Data
|01.01.2007 |
//+-----
---+
//| Descr. | Rotina para validar a linha de dados.
|
//+-----
---+
//| Uso      | Para treinamento e capacitação.
|
//+-----
---+
User Function Mod2LOk()
Local lRet := .T.

Local cMensagem := "Não será permitido linhas sem o centro de
custo."
If !aCOLS[n, Len(aHeader)+1]
    If Empty(aCOLS[n,GdFieldPos("ZA3_CCUSTO")])
        MsgBox(cMensagem,cCadastro)
        lRet := .F.
    Endif
Endif
Return( lRet )

```

#### **Função auxiliar: Validação do código do centro de custo para todas as linhas**

---

```

//+-----
---+
//| Rotina | Mod2TOk | Autor | Robson Luiz (rleg)    | Data
|01.01.2007 |
//+-----
---+
//| Descr. | Rotina para validar toda as linhas de dados.
|
//+-----
---+
//| Uso      | Para treinamento e capacitação.
|
//+-----
---+
User Function Mod2TOk()
    Local lRet := .T.
    Local nI := 0
    Local cMensagem := "Não será permitido linhas sem o centro de
custo."

```

```
For nI := 1 To Len( aCOLS )
    If aCOLS[nI, Len(aHeader)+1]
        Loop
    Endif
    If !aCOLS[nI, Len(aHeader)+1]
        If Empty(aCOLS[n, GdFieldPos("ZA3_CCUSTO")])
            MsgAlert(cMensagem,cCadastro)
            lRet := .F.
            Exit
        Endif
    Endif
    Next nI
Return( lRet )
```

### **2.11.3 Função Modelo2()**

A função Modelo2() é uma interface pré-definida pela Microsiga que implementa de forma padronizada os componentes necessários à manipulação de estruturas de dados nas quais o cabeçalho e os itens da informação compartilham o mesmo registro físico.

Seu objetivo é atuar como um facilitador de codificação, permitindo a utilização dos recursos básicos dos seguintes componentes visuais:

- **MsDialog()**
- **TGet()**
- **TSay()**
- **MsNewGetDados()**
- **EnchoiceBar()**



- A função Modelo2() não implementa as regras de visualização, inclusão, alteração e exclusão, como uma AxCadastro() ou AxFunction().
- A inicialização das variáveis Private utilizada nos cabeçalhos e rodapés, bem como a inicialização e gravação do aCols devem ser realizadas pela rotina que “suporta” a execução da Modelo2().
- Da mesma forma, o Browse deve ser tratado por esta rotina, sendo comum a função Modelo2() estar vinculada ao uso de uma função MBrowse().

**Sintaxe:** **Modelo2([cTitulo], [aCab], [aRoda], [aGrid], [nOpc], [cLinhaOk], [cTudoOk])**

**Parâmetros:**

<b>cTitulo</b>	Título da janela
<b>aCab</b>	Array contendo as informações que serão exibidas no cabeçalho na forma de Enchoice() aCab[n][1] (Caractere) := Nome da variável private que será vinculada ao campo da Enchoice(). aCab[n][2] (Array) := Array com as coordenadas do campo na tela {Linha, Coluna} aCab[n][3] (Caractere) := Título do campo na tela aCab[n][4] (Caractere) := Picture de formatação do get() do campo. aCab[n][5] (Caractere) := Função de validação do get() do campo. aCab[n][6] (Caractere) := Nome da consulta padrão que será executada para o campo via tecla F3 aCab[n][7] (Lógico) := Se o campo estará livre para digitação.

<b>aRoda</b>	Array contendo as informações que serão exibidas no cabeçalho na forma de Enchoice(), no mesmo formato que o aCab.
<b>aGrid</b>	Array contendo as coordenadas da GetDados() na tela. Padrão := {44,5,118,315}
<b>nOpc</b>	Opção selecionada na função MBrowse, ou que deseja ser passada para controle da Modelo2, aonde: 2 – Visualizar 3 - Incluir 4 - Alterar 5 - Excluir
<b>cLinhaOk</b>	Função para validação da linha na GetDados()
<b>cTudoOk</b>	Função para validação na confirmação da tela de interface da função Modelo2().

**Retorno:**

<b>Lógico</b>	Indica se a tela da interface Modelo2() foi confirmada ou cancelada pelo usuário.
---------------	---

---

**Exemplo: Utilização da Modelo2() para visualização do Cadastro de Tabelas (SX5)**

---

```
#include "protheus.ch"

//+-----
---+
//| Rotina | MBRW2SX5| Autor | ARNALDO RAYMUNDO JR. | Data
|01.01.2007 |
//+-----
---+
//| Descr. | UTILIZACAO DA MODELO2() PARA VISUALIZAÇÃO DO SX5.
|
//+-----
---+
//| Uso      | CURSO DE ADVPL
|
//+-----
---+
USER FUNCTION MBrw2Sx5()

Local cAlias           := "SX5"

Private cCadastro       := "Arquivo de Tabelas"
Private aRotina    := {}
Private cDelFunc        := ".T." // Validação para a exclusão. Pode-
se utilizar ExecBlock

AADD(aRotina,{"Pesquisar" , "AxPesqui" ,0,1})
```

```

AADD(aRotina,{"Visualizar" , "U_SX52Vis" ,0,2})
AADD(aRotina,{"Incluir" , "U_SX52Inc" ,0,3})
AADD(aRotina,{"Alterar" , "U_SX52Alt" ,0,4})
AADD(aRotina,{"Excluir" , "U_SX52Exc" ,0,5})

dbSelectArea(cAlias)
dbSetOrder(1)
mBrowse( 6,1,22,75,cAlias)

Return

USER FUNCTION SX52INC(cAlias,nReg,nOpc)

//Local nUsado      := 0
Local cTitulo       := "Inclusao de itens - Arquivo de Tabelas"
Local aCab          := {} // Array com descricao dos campos do
Cabecalho do Modelo 2
Local aRoda          := {} // Array com descricao dos campos do
Rodape do Modelo 2
Local aGrid          := {80,005,050,300} //Array com coordenadas
da GetDados no modelo2 - Padrao: {44,5,118,315}
                                         // Linha Inicial - Coluna Inicial - +Qts
Linhas - +Qts Colunas : {080,005,050,300}
Local cLinhaOk      := "AllwaysTrue()" // Validacoes na linha da
GetDados da Modelo 2
Local cTudoOk        := "AllwaysTrue()" // Validacao geral da GetDados
da Modelo 2
Local lRetMod2      := .F. // Retorno da função Modelo2 - .T. Confirmou
/ .F. Cancelou
Local nColuna        := 0

// Variaveis para GetDados()
Private aCols         := {}
Private aHeader := {}

// Variaveis para campos da Enchoice()
Private cX5Filial := xFilial("SX5")
Private cX5Tabela := SPACE(5)

// Montagem do array de cabeçalho
// AADD(aCab,{"Variável" ,{[L,C]
,"Título","Picture","Valid","F3",lEnable}}
AADD(aCab,{"cX5Filial" ,{015,010} , "Filial","@!",,,,F.})
AADD(aCab,{"cX5Tabela" ,{015,080} , "Tabela","@!",,,,T.})

// Montagem do aHeader
AADD(aHeader,{"Chave" , "X5_CHAVE","@!",5,0,"AllwaysTrue()",;
               "", "C", "", "R"})
AADD(aHeader,{"Descricao" ,
               "X5_DESCRI","@!",40,0,"AllwaysTrue()",;

```

```

        "", "C", "", "R"})}

// Montagem do aCols
aCols := Array(1,Len(aHeader)+1)

// Inicialização do aCols
For nColuna := 1 to Len(aHeader)

    If aHeader[nColuna][8] == "C"
        aCols[1][nColuna] := SPACE(aHeader[nColuna][4])
    Elseif aHeader[nColuna][8] == "N"
        aCols[1][nColuna] := 0
    Elseif aHeader[nColuna][8] == "D"
        aCols[1][nColuna] := CTOD("")
    Elseif aHeader[nColuna][8] == "L"
        aCols[1][nColuna] := .F.
    Elseif aHeader[nColuna][8] == "M"
        aCols[1][nColuna] := ""
    Endif

Next nColuna

aCols[1][Len(aHeader)+1] := .F. // Linha não deletada
lRetMod2 := Modelo2(cTitulo,aCab,aRoda,aGrid,nOpc,cLinhaOk,cTudoOk)

IF lRetMod2
    //MsgInfo("Você confirmou a operação","MBRW2SX5")
    For nLinha := 1 to len(aCols)
        // Campos de Cabeçalho
        Reclock("SX5",.T.)
        SX5->X5_FILIAL := cX5Filial
        SX5->X5_TABELA := cX5Tabela
        // Campos do aCols
        //SX5->X5_CHAVE := aCols[nLinha][1]
        //SX5->X5_DESCRI := aCols[nLinha][2]
        For nColuna := 1 to Len(aHeader)
            SX5->&(aHeader[nColuna][2]) :=
aCols[nLinha][nColuna]
        Next nColuna
        MsUnLock()
    Next nLinha
ELSE
    MsgAlert("Você cancelou a operação","MBRW2SX5")
ENDIF
Return

```



## Exercícios

### Exercício 08

Implementar uma Modelo2() para a tabela padrão do ERP – SX5: Arquivo de Tabelas

### Exercício 09

Implementar as funções de Visualização, Alteração e Exclusão para a Modelo2 desenvolvida para o SX5.

### Exercício 10

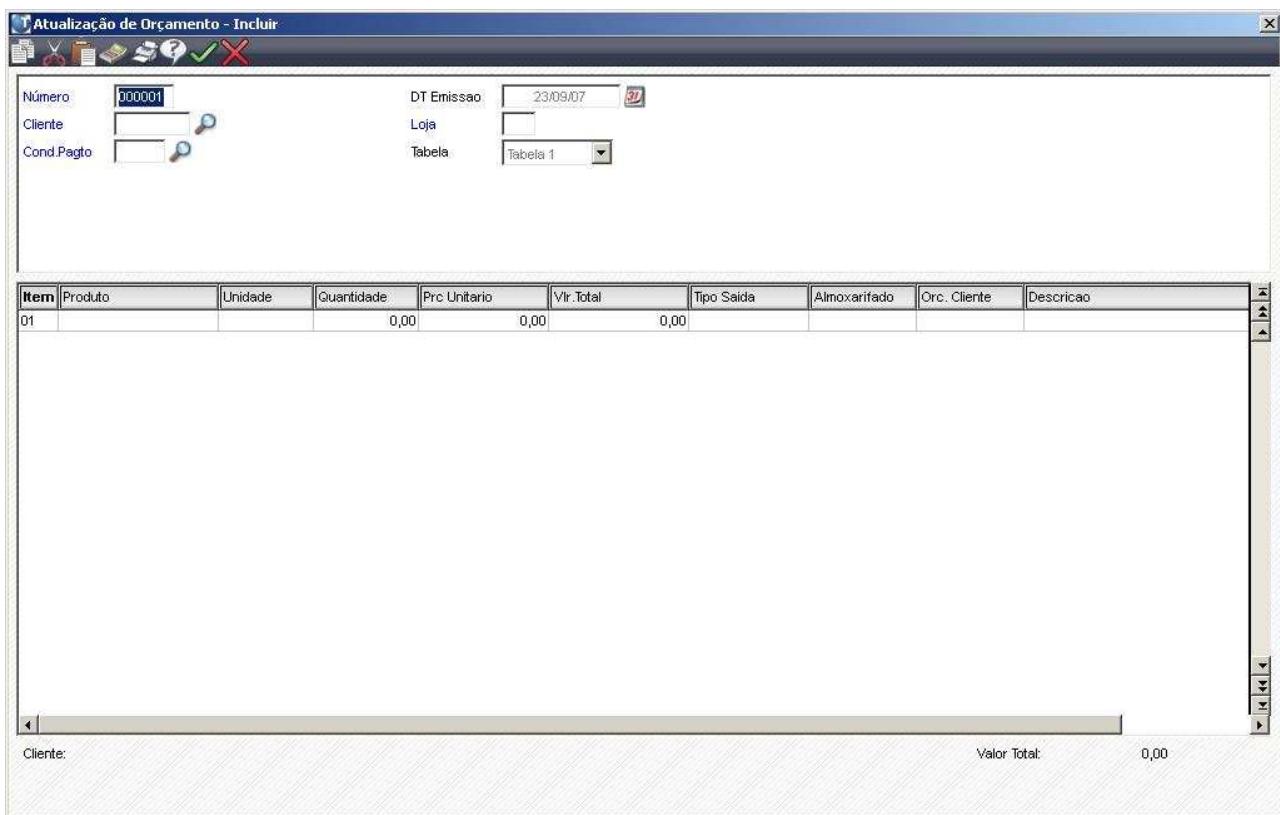
Implementar uma Modelo2() para a tabela padrão do ERP – SB1: Tabela de Produtos

## **2.12    Modelo3()**

O nome Modelo 3, assim como a Modelo 2 foi conceituado pela Microsiga por se tratar de um protótipo de tela para entrada de dados. Inicialmente, vamos desmistificar dois pontos:

- **Função Modelo3()** – Trata-se de uma função pronta que contempla o protótipo Modelo 3, porém, este é um assunto que não iremos tratar aqui, visto que é uma funcionalidade simples que quando necessário intervir em algo na rotina não há muito recurso para tal.
- **Protótipo Modelo 3** – Trata-se de uma tela, como a figura abaixo, onde seu objetivo é efetuar a manutenção em vários registros de uma só vez relacionada a outro registro de outra tabela, ou seja, aqui teremos o relacionamento de registros “pai e filho”. Então, é preciso se preocupar com este relacionamento. Por exemplo: efetuar a manutenção em um pedido de vendas, onde terá um registro em uma tabela referente à cabeça do pedido, e, outra tabela, com os registros referentes aos itens deste pedido de vendas.

Para ganharmos tempo não será apresentada aqui toda a explicação e as montagens para a função **EnchoiceBar**, comando **MsDialog**, **Say** e **MsGet** e para os vetores **aHeader** e **aCOLS**. Entretanto, todos estes estarão na codificação do código fonte. A figura abaixo mostra exatamente o que é a tela protótipo Modelo 3:



**Figura: Protótipo Modelo 3**

Este protótipo é constituído de MsDialog, EnchoiceBar, Enchoice, MsGetDados, Say e Get.

Diante dos expostos até o momento houve um novo nome para nós, é ele a função Enchoice, o que é?

#### **Função Enchoice() – Objeto MsMGet()**

A função Enchoice ou o objeto MsMGet são recursos baseados no dicionário de dados para verificar campos obrigatórios, validações, gatilhos, consulta padrão e etc. Assim também para criar pastas de cadastros. Estes podem ser usados tanto com variáveis de memórias com o escopo Private como diretamente os campos da tabela que se refere. A diferença entre a função Enchoice e o objeto MsMGet é que a função não retorna o nome da variável de objeto exportável criado.

A estrutura para montar um programa com o protótipo modelo 3 é semelhante ao protótipo modelo 2, porém a diferença real é a utilização da função Enchoice ou o objeto MsMGet, para este documento iremos trabalhar com a função.

**Sintaxe: Enchoice( cAlias, nReg, nOpc, aAc, cOpc, cTextExclui, aAcho, aPos, aCpos, nNum, nColMens, cMensagem, cTudOk, oObj, lVirtual)**

**Parâmetros:**

<b>cAlias</b>	Alias do dados a serem cadastrados.
<b>nReg</b>	Número do registro da tabela a ser editado.
<b>uPar1</b>	Parâmetro reservado.
<b>uPar2</b>	Parâmetro reservado.
<b>uPar3</b>	Parâmetro reservado.
<b>aAcho</b>	Vetor com os campos que serão apresentados pela MsMGet.
<b>aPos</b>	Vetor com as coordenadas onde a MsMGet será criada no formato {coord. superior, coord. esquerda, coord. direita, coord. inferior}. Função executada para validar o contexto da linha atual do aCols.
<b>aCpos</b>	Vetor com os campos que poderão ser alterados.
<b>uPar4</b>	Parâmetro reservado. Nome dos campos do tipo caracter que utilizarão incremento automático. Este parâmetro deve ser no formato “+<nome do primeiro campo>+<nome do segundo campo>+...”.
<b>uPar5</b>	Parâmetro reservado.
<b>uPar6</b>	Parâmetro reservado.
<b>uPar7</b>	Parâmetro reservado.
<b>oWnd</b>	Objeto no qual a MsMGet será criada.
<b>uPar8</b>	Parâmetro reservado.
<b>IMemoria</b>	Indica se serão usadas variáveis de memória ou os campos da tabela para cadastramento dos dados. Valor padrão falso.
<b>IColuna</b>	Indica se a MsMGet será apresentada com um objeto por linha (uma coluna). Valor padrão falso. Parâmetro reservado.
<b>uPar9</b>	Parâmetro reservado.
<b>ISemPastas</b>	Indica se não serão usadas as Pastas de Cadastro na MsMGet. Função executada para validar a exclusão de uma linha do aCols.

Vale lembrar que nós programadores reproveitamos muito o que já existe, isto é para simplesmente ganhamos tempo, e no caso da utilização da função Enchoice é preciso criar as variáveis de memórias que levam o mesmo nome dos campos da tabela em questão. Por exemplo, o campo A2\_NOME da tabela SA2 (cadastro de fornecedores) quando queremos referenciar o campo, usa-se o prefixo da tabela e o campo em questão, desta forma:

SA2->A2\_NOME

Agora quando queremos referenciar a uma variável que está com o conteúdo do mesmo campo criamos outro recurso, desta forma:

M->A2\_NOME

E para criar variáveis com o nome do campo utilizamos um código de bloco (code-block) e mais um laço de leitura para atribuir valores iniciais a cada uma delas. Então, o procedimento é o seguinte:

```
Private bCampo := { |nField| Field(nField) }
```

E em outro momento, aproveitamos a variável bCampo para facilitar a atribuição. Veja o exemplo abaixo :

```
For nX := 1 To FCount()
M->&( Eval( bCampo, nX ) ) := Atribuição inicial ou atribuição de
valor
Next nX
```

Ou seja, fazer para todos os campos, e a cada campo criar a variável com a atribuição inicial ou atribuição de valor.

## **2.12.1 Estrutura de um programa utilizando a Modelo3()**

O exemplo abaixo demonstra a montagem de um programa para a utilização do protótipo Modelo 3. Antes de iniciarmos o exemplo, vamos estruturar o programa.

### **Estrutura do programa**

#### **Linhas Programa**

```
1  Função principal;
2      Declaração e atribuição de variáveis;
3      Acesso à tabela principal e sua ordem;
4      Chamada da função MBrowse;
5  Fim da função principal.
6
7  Função de visualização, alteração e exclusão;
8      Declaração e atribuição de variáveis;
9      Acesso ao primeiro registro da chave em que está posicionado na
MBrowse;
10     Construção das variáveis de memória M->???;
11     Montagem do vetor aHeader por meio do Dicionário de Dados;
12     Montagem do vetor aCOLS de todos os registros referente a chave
principal em que está posicionado na MBrowse;
13     Instância da MsDialog;
14     Execução da função Enchoice;
15     Instância do objeto MsGetDados;
16     Ativar o objeto principal que é o objeto da janela;
17     Se for operação diferente de visualização e clicou no botão OK;
18         A operação é de Alteração?
19             Chamar a função para alterar os dados;
20         Caso contrário:
21             Chamar a função para excluir os dados;
22  Fim da função de visualização, alteração e exclusão.
23
24  Função de inclusão;
25      Declaração e atribuição de variáveis;
26      Construção das variáveis de memória M->???;
27      Montagem do vetor aHeader por meio do dicionário de dados;
28      Montagem do vetor aCOLS com o seu conteúdo conforme o
inicializador padrão do campo ou vazio, pois trata-se de uma inclusão;
29      Instância da MsDialog;
30      Instância dos objetos TSay e TGet;
31      Instância do objeto MsGetDados;
32      Ativar o objeto principal que é o objeto da janela;
33      Se clicou no botão OK;
34          Chamar a função para incluir os dados;
35  Fim da função de inclusão.
```

## **Rotina principal**

---

```
//+-----  
---+  
//| Rotina | xModelo3 | Autor | Robson Luiz (rleg) | Data |  
01.01.2007 |  
//+-----  
---+  
//| Descr. | Função exemplo do protótipo Modelo3.  
|  
//+-----  
---+  
//| Uso     | Para treinamento e capacitação.  
|  
//+-----  
---+  
#Include "Protheus.ch"  
  
User Function xModelo3()  
Private cCadastro := "Protótipo Modelo 3"  
Private aRotina := {}  
Private oCliente  
Private oTotal  
Private cCliente := ""  
Private nTotal := 0  
  
Private bCampo := {|nField| FieldName(nField) }  
  
Private aSize := {}  
Private aInfo := {}  
Private aObj := {}  
Private aPObj := {}  
Private aPGet := {}  
  
// Retorna a área útil das janelas Protheus  
aSize := MsAdvSize()  
  
// Será utilizado três áreas na janela  
// 1ª - Enchoice, sendo 80 pontos pixel  
// 2ª - MsGetDados, o que sobrar em pontos pixel é para este objeto  
// 3ª - Rodapé que é a própria janela, sendo 15 pontos pixel  
AADD( aObj, { 100, 080, .T., .F. })  
AADD( aObj, { 100, 100, .T., .T. })  
AADD( aObj, { 100, 015, .T., .F. })  
  
// Cálculo automático da dimensões dos objetos (altura/largura) em  
pixel  
aInfo := { aSize[1], aSize[2], aSize[3], aSize[4], 3, 3 }  
aPObj := MsObjSize( aInfo, aObj )
```

```

// Cálculo automático de dimensões dos objetos MSGET
aPGet := MsObjGetPos( (aSize[3] - aSize[1]), 315, { {004, 024, 240,
270} } )

AADD( aRotina, {"Pesquisar" , "AxPesqui" ,0,1})
AADD( aRotina, {"Visualizar" , 'U_Mod3Mnt',0,2})
AADD( aRotina, {"Incluir"      , 'U_Mod3Inc',0,3})
AADD( aRotina, {"Alterar"     , 'U_Mod3Mnt',0,4})
AADD( aRotina, {"Excluir"     , 'U_Mod3Mnt',0,5})

dbSelectArea("ZA1")
dbSetOrder(1)
dbGoTop()
MBrowse(,,,,"ZA1")
Return

```

## Função de Inclusão

---

```

//+-----
---+
//| Rotina | Mod3Inc | Autor | Robson Luiz (rleg) | Data |
01.01.2007 |
//+-----
---+
//| Descr. | Rotina para incluir dados.
|
//+-----
---+
//| Uso      | Para treinamento e capacitação.
|
//+-----
---+
User Function Mod3Inc( cAlias, nReg, nOpc )
Local oDlg
Local oGet
Local nX := 0
Local nOpcA := 0

Private aHeader := {}
Private aCOLS := {}
Private aGets := {}
Private aTela := {}

dbSelectArea( cAlias )
dbSetOrder(1)

For nX := 1 To FCount()
    M->&( Eval( bCampo, nX ) ) := CriaVar( FieldName( nX ), .T. )

```

```

Next nX

Mod3aHeader()
Mod3aCOLS( nOpc )

DEFINE MSDIALOG oDlg TITLE cCadastro FROM ;
aSize[7],aSize[1] TO aSize[6],aSize[5] OF oMainWnd PIXEL
    EnChoice( cAlias, nReg, nOpc, , , , aPObj[1] )

    // Atualização do nome do cliente
    @ aPObj[3,1],aPGet[1,1] SAY "Cliente: " SIZE 70,7 OF oDlg PIXEL
    @ aPObj[3,1],aPGet[1,2] SAY oCliente VAR cCliente SIZE 98,7 OF
oDlg PIXEL

    // Atualização do total
    @ aPObj[3,1],aPGet[1,3] SAY "Valor Total: "   SIZE 70,7 OF oDlg
PIXEL
    @ aPObj[3,1],aPGet[1,4] SAY oTotal VAR nTotal ;
PICT "@E 9,999,999,999.99" SIZE 70,7 OF oDlg PIXEL

    oGet :=
MSGGetDados():New(aPObj[2,1],aPObj[2,2],aPObj[2,3],aPObj[2,4],;
nOpc,"U_Mod3LOk()",".T.", "+ZA2_ITEM",.T.)

ACTIVATE MSDIALOG oDlg ON INIT EnchoiceBar(oDlg,;
{|| IIF( Mod3TOk().And.Obrigatorio( aGets, aTela ), ( nOpcA := 1,
oDlg:End() ), NIL );;;
{|| oDlg:End() }}

If nOpcA == 1 .And. nOpc == 3
    Mod3Grv( nOpc )
    ConfirmSXE()
Endif
Return

```

### **Função de Visualização, Alteração e Exclusão**

---

```

//+-----
---+
//| Rotina | Mod3Mnt | Autor | Robson Luiz (rleg)  | Data |
01.01.2007 |
//+-----
---+
//| Descr. | Rotina para Visualizar, Alterar e Excluir dados.
|
//+-----
---+
//| Uso     | Para treinamento e capacitação.
|

```

```

//+-----
---+
User Function Mod3Mnt( cAlias, nReg, nOpc )
Local oDlg
Local oGet
Local nX := 0
Local nOpcA := 0
Private aHeader := {}
Private aCOLS := {}
Private aGets := {}
Private aTela := {}
Private aREG := {}

dbSelectArea( cAlias )
dbSetOrder(1)

For nX := 1 To FCount()
    M->&( Eval( bCampo, nX ) ) := FieldGet( nX )
Next nX

Mod3aHeader()
Mod3aCOLS( nOpc )
DEFINE MSDIALOG oDlg TITLE cCadastro FROM ;
aSize[7],aSize[1] TO aSize[6],aSize[5] OF oMainWnd PIXEL
EnChoice( cAlias, nReg, nOpc, , , , aPObj[1])

    // Atualização do nome do cliente
    @ aPObj[3,1],aPGet[1,1] SAY "Cliente: " SIZE 70,7 OF oDlg PIXEL
    @ aPObj[3,1],aPGet[1,2] SAY oCliente VAR cCliente SIZE 98,7 OF
oDlg PIXEL

    // Atualização do total
    @ aPObj[3,1],aPGet[1,3] SAY "Valor Total: " SIZE 70,7 OF oDlg
PIXEL
    @ aPObj[3,1],aPGet[1,4] SAY oTotal VAR nTotal PICTURE ;
"@E 9,999,999,999.99" SIZE 70,7 OF oDlg PIXEL

    U_Mod3Cli()

    oGet :=
MSGDados():New(aPObj[2,1],aPObj[2,2],aPObj[2,3],aPObj[2,4],;
nOpc,"U_Mod3LOk()",".T.","+ZA2_ITEM",.T.)

ACTIVATE MSDIALOG oDlg ON INIT EnchoiceBar(oDlg,;
{|| IIF( Mod3TOk().And.Obrigatorio( aGets, aTela ), ( nOpcA := 1,
oDlg:End() ), NIL ) },;
{|| oDlg:End() })

If nOpcA == 1 .And. ( nOpc == 4 .Or. nOpc == 5 )
Mod3Grv( nOpc, aREG )

```

```
Endif  
Return
```

### Função para montar o vetor aHeader

```
//+-----  
---+  
//| Rotina | Mod3aHeader | Autor | Robson Luiz (rleg)  
|Data|01.01.2007 |  
//+-----  
---+  
//| Descr. | Rotina para montar o vetor aHeader.  
|  
//+-----  
---+  
//| Uso     | Para treinamento e capacitação.  
|  
//+-----  
---+  
Static Function Mod3aHeader()  
Local aArea := GetArea()  
  
dbSelectArea("SX3")  
dbSetOrder(1)  
dbSeek("ZA2")  
While !EOF() .And. X3_ARQUIVO == "ZA2"  
    If X3Uso(X3_USADO) .And. cNivel >= X3_NIVEL  
        AADD( aHeader, { Trim( X3Titulo() ),;  
                      X3_CAMPO,;  
                      X3_PICTURE,;  
                      X3_TAMANHO,;  
                      X3_DECIMAL,;  
                      X3_VALID,;  
                      X3_USADO,;  
                      X3_TIPO,;  
                      X3_ARQUIVO,;  
                      X3_CONTEXT} )  
    Endif  
    dbSkip()  
End  
RestArea(aArea)  
Return
```

### Função para montar o vetor aCols

```
//+-----  
---+
```

```

//| Rotina | Mod3aCOLS | Autor | Robson Luiz (rleg) | Data |
01.01.2007 |
//+-----
---+
//| Descr. | Rotina para montar o vetor aCOLS.
|
//+-----
---+
//| Uso      | Para treinamento e capacitação.
|
//+-----
---+
Static Function Mod3aCOLS( nOpc )
Local aArea := GetArea()
Local cChave := ""
Local cAlias := "ZA2"
Local nl := 0

If nOpc <> 3
    cChave := ZA1->ZA1_NUM

        dbSelectArea( cAlias )
        dbSetOrder(1)
        dbSeek( xFilial( cAlias ) + cChave )
        While !EOF() .And. ZA2->( ZA2_FILIAL + ZA2_NUM ) == xFilial(
cAlias ) + cChave
            AADD( aREG, ZA2->( RecNo() ) )
            AADD( aCOLS, Array( Len( aHeader ) + 1 ) )
            For nl := 1 To Len( aHeader )
                If aHeader[nl,10] == "V"
                    aCOLS[Len(aCOLS),nl] :=
CriaVar(aHeader[nl,2],.T.)
                Else
                    aCOLS[Len(aCOLS),nl] :=
FieldGet(FieldPos(aHeader[nl,2]))
                Endif
                Next nl
                aCOLS[Len(aCOLS),Len(aHeader)+1] := .F.
                dbSkip()
            End
        Else
            AADD( aCOLS, Array( Len( aHeader ) + 1 ) )
            For nl := 1 To Len( aHeader )
                aCOLS[1, nl] := CriaVar( aHeader[nl, 2], .T. )
            Next nl
            aCOLS[1, GdFieldPos("ZA2_ITEM")] := "01"
            aCOLS[1, Len( aHeader )+1 ] := .F.
        Endif
        Restarea( aArea )
    Return

```

## Função para atribuir o nome do cliente a variável

```
//+-----  
---+  
//| Rotina | Mod3Cli | Autor | Robson Luiz (rleg) | Data |  
01.01.2007 |  
//+-----  
---+  
//| Descr. | Rotina para atualizar a variável com o nome do cliente.  
|  
//+-----  
---+  
//| Uso     | Para treinamento e capacitação.  
|  
//+-----  
---+  
  
User Function Mod3Cli()  
cCliente := Posicione( "SA1", 1, xFilial("SA1") + M->(ZA1_CLIENT +  
ZA1_LOJA), "A1_NREDUZ" )  
oCliente:Refresh()  
Return(.T.)
```

## Função para validar a mudança de linha na MsGetDados()

---

```
//+-----  
---+  
//| Rotina | Mod3LOk | Autor | Robson Luiz (rleg) | Data |  
01.01.2007 |  
//+-----  
---+  
//| Descr. | Rotina para atualizar a variável com o total dos itens.  
|  
//+-----  
---+  
//| Uso     | Para treinamento e capacitação.  
|  
//+-----  
---+  
User Function Mod3LOk()  
Local nl := 0  
nTotal := 0  
For nl := 1 To Len( aCOLS )  
    If aCOLS[nl,Len(aHeader)+1]  
        Loop  
    Endif  
    nTotal+=Round(aCOLS[nl,GdFieldPos("ZA2_QTDVEN")]*;  
        aCOLS[nl,GdFieldPos("ZA2_PRCVEN")],2)  
Next nl  
oTotal:Refresh()  
Return(.T.)
```

## Função para validar se todas as linhas estão preenchidas

---

```
//+-----  
---+  
//| Rotina | Mod3TOk | Autor | Robson Luiz (rleg) | Data |  
01.01.2007 |  
//+-----  
---+  
//| Descr. | Rotina para validar os itens se foram preenchidos.  
|  
//+-----  
---+  
//| Uso     | Para treinamento e capacitação.  
|  
//+-----  
---+  
Static Function Mod3TOk()  
Local nl := 0  
Local lRet := .T.
```

```

For nI := 1 To Len(aCOLS)
    If aCOLS[nI, Len(aHeader)+1]
        Loop
    Endif
    If Empty(aCOLS[nI, GdFieldPos("ZA2_PRODUT")]) .And. !Ret
        MsgAlert("Campo PRODUTO preenchimento obrigatorio",cCadastro)
        !Ret := .F.
    Endif
    If Empty(aCOLS[nI, GdFieldPos("ZA2_QTDVEN")]) .And. !Ret
        MsgAlert("Campo QUANTIDADE preenchimento
obrigatorio",cCadastro)
        !Ret := .F.
    Endif
    If Empty(aCOLS[nI, GdFieldPos("ZA2_PRCVEN")]) .And. !Ret
        MsgAlert("Campo PRECO UNITARIO preenchimento
obrigatorio",cCadastro)
        !Ret := .F.
    Endif

    If !Ret
        Exit
    Endif
Next i
Return( !Ret )

```

```

//+-----
---+
//| Rotina | Mod3Grv | Autor | Robson Luiz (rleg) | Data |
01.01.2007 |
//+-----
---+
//| Descr. | Rotina para efetuar a gravação nas tabelas.
|
//+-----
---+
//| Uso      | Para treinamento e capacitação.
|
//+-----
---+
Static Function Mod3Grv( nOpc, aAltera )
Local nX := 0
Local nI := 0

// Se for inclusão
If nOpc == 3
    // Grava os itens
    dbSelectArea("ZA2")
    dbSetOrder(1)
    For nX := 1 To Len( aCOLS )

```

```

        If !aCOLS[ nX, Len( aCOLS ) + 1 ]
            RecLock( "ZA2", .T. )
            For nl := 1 To Len( aHeader )
                FieldPut( FieldPos( Trim( aHeader[nl, 2] ) ), 
aCOLS[nX,nl] )
            Next nl
            ZA2->ZA2_FILIAL := xFilial("ZA2")
            ZA2->ZA2_NUM := M->ZA1_NUM
            MsUnLock()
        Endif
    Next nX

    // Grava o Cabeçalho
    dbSelectArea( "ZA1" )
    RecLock( "ZA1", .T. )
    For nX := 1 To FCount()
        If "FILIAL" $ FieldName( nX )
            FieldPut( nX, xFilial( "ZA1" ) )
        Else
            FieldPut( nX, M->&( Eval( bCampo, nX ) ) )
        Endif
    Next nX
    MsUnLock()
Endif

// Se for alteração
If nOpc == 4
    // Grava os itens conforme as alterações
    dbSelectArea("ZA2")
    dbSetOrder(1)
    For nX := 1 To Len( aCOLS )
        If nX <= Len( aREG )
            dbGoto( aREG[nX] )
            RecLock("ZA2",.F.)
            If aCOLS[ nX, Len( aHeader ) + 1 ]
                dbDelete()
            Endif
        Else
            If !aCOLS[ nX, Len( aHeader ) + 1 ]
                RecLock( "ZA2", .T. )
            Endif
        Endif
    Endif

    If !aCOLS[ nX, Len(aHeader)+1 ]
        For nl := 1 To Len( aHeader )
            FieldPut( FieldPos( Trim( aHeader[ nl, 2] ) 
aCOLS[ nX, nl ] ) )
    );

```

```

        Next nl
        ZA2->ZA2_FILIAL := xFilial("ZA2")
        ZA2->ZA2_NUM := M->ZA1_NUM
    Endif
    MsUnLock()

Next nX

// Grava o Cabeçalho
dbSelectArea("ZA1")
RecLock( "ZA1", .F. )
For nX := 1 To FCount()
    If "FILIAL" $ FieldName( nX )
        FieldPut( nX, xFilial("ZA1"))
    Else
        FieldPut( nX, M->&( Eval( bCampo, nX ) ) )
    Endif
Next
MsUnLock()
Endif

// Se for exclusão
If nOpc == 5
    // Deleta os Itens
    dbSelectArea("ZA2")
    dbSetOrder(1)
    dbSeek(xFilial("ZA2") + M->ZA1_NUM)
    While !EOF() .And. ZA2->(ZA2_FILIAL + ZA2_NUM) ==
xFilial("ZA2") +
        M->ZA1_NUM
        RecLock("ZA2")
        dbDelete()
        MsUnLock()
        dbSkip()
End

// Deleta o Cabeçalho
dbSelectArea("ZA1")
RecLock("ZA1",.F.)
dbDelete()
MsUnLock()
Endif
Return

```

## **2.12.2 Função Modelo3()**

A função Modelo3() é uma interface pré-definida pela Microsiga que implementa de forma padronizada os componentes necessários e a manipulação de estruturas de dados nas quais o cabeçalho e os itens da informação estão em tabelas separadas.

Seu objetivo é atuar como um facilitador de codificação, permitindo a utilização dos recursos básicos dos seguintes componentes visuais:

- **MsDialog()**
- **Enchoice()**
- **EnchoiceBar()**
- **MsNewGetDados()**



*Importante*

- A função Modelo3() não implementa as regras de visualização, inclusão, alteração e exclusão, como uma AxCadastro() ou AxFunction().
- A inicialização dos campos utilizados na Enchoice() deve ser realizadas pela rotina que “suporta” a execução da Modelo3(), normalmente através do uso da função RegToMemory().
- Da mesma forma, o Browse deve ser tratado por esta rotina, sendo comum a Modelo3() estar vinculada ao uso de uma MBrowse().

**Sintaxe: Modelo3 ([cTitulo], [cAliasE], [cAliasGetD], [aCposE], [cLinOk],  
[cTudOk], [nOpcE], [nOpcG], [cFieldOk])**

**Parâmetros:**

<b>cTitulo</b>	Título da janela.
<b>cAliasE</b>	Alias da tabela que será utilizada na Enchoice.
<b>cAliasGetD</b>	Alias da tabela que será utilizada na GetDados.
<b>aCposE</b>	Nome dos campos, pertencentes ao Alias especificado o parâmetro cAliasE, que deverão ser exibidos na Enchoice: AADD(aCposE, {"nome_campo"}).
<b>cLinhaOk</b>	Função para validação da linha na GetDados().
<b>cTudoOk</b>	Função para validação na confirmação da tela de interface da Modelo2().
<b>nOpcE</b>	Opção selecionada na função MBrowse, ou que deseja ser passada para controle da função Enchoice da função Modelo3, aonde: 2 – Visualizar 3 - Incluir 4 - Alterar 5 - Excluir
<b>nOpcG</b>	Opção selecionada na função MBrowse, ou que deva ser passada para controle da função GetDados da função Modelo3, aonde: 2 – Visualizar

	3 - Incluir 4 - Alterar 5 - Excluir
<b>cFieldOk</b>	Validação dos campos da função Enchoice()

**Retorno:**

<b>Lógico</b>	Indica se a tela da interface Modelo2() foi confirmada ou cancelada pelo usuário.
---------------	---

**Exemplo: Utilização da Modelo3() para Pedidos de Vendas (SC5,SC6)**

```
#INCLUDE "protheus.ch"

//+-----
---+
//| Rotina | MBRWMOD3| Autor | ARNALDO RAYMUNDO JR. |Data |
01.01.2007 |
//+-----
---+
//| Descr. | EXEMPLO DE UTILIZACAO DA MODELO3().
|
//+-----
---+
//| Uso      | CURSO DE ADVPL
|
//+-----
---+
User Function MbrwMod3()

Private cCadastro          := "Pedidos de Venda"
Private aRotina   := {}
Private cDelFunc           := ".T." // Validação para a exclusão. Pode-
se utilizar ExecBlock
Private cAlias              := "SC5"

AADD(aRotina,{ "Pesquisa","AxPesqui"           ,,0,1})
AADD(aRotina,{ "Visual"  ,,"U_Mod3All"         ,,0,2})
AADD(aRotina,{ "Inclui"    ,,"U_Mod3All"         ,,0,3})
AADD(aRotina,{ "Altera"   ,,"U_Mod3All"         ,,0,4})
AADD(aRotina,{ "Exclui"   ,,"U_Mod3All"         ,,0,5})

dbSelectArea(cAlias)
dbSetOrder(1)
mBrowse( 6,1,22,75,cAlias)

Return
```

```
User Function Mod3All(cAlias,nReg,nOpcx)
```

```
Local cTitulo := "Cadastro de Pedidos de Venda"
Local cAliasE := "SC5"
Local cAliasG := "SC6"
Local cLinOk := "AlwaysTrue()"
Local cTudOk := "AlwaysTrue()"
Local cFieldOk:= "AlwaysTrue()"
Local aCposE := {}
Local nUsado, nX := 0

// Última versão da função Mod3All

//³ Opções de acesso para a Modelo 3
// Acesso ao Banco de Dados

Do Case
    Case nOpcx==3; nOpcE:=3 ; nOpcG:=3      // 3 - "INCLUIR"
    Case nOpcx==4; nOpcE:=3 ; nOpcG:=3      // 4 - "ALTERAR"
    Case nOpcx==2; nOpcE:=2 ; nOpcG:=2      // 2 - "VISUALIZAR"
    Case nOpcx==5; nOpcE:=2 ; nOpcG:=2      // 5 - "EXCLUIR"
EndCase

// Cria variáveis M->????? da Enchoice
// RegToMemory("SC5", (nOpcx==3 .or. nOpcx==4 )) // Se for inclusão ou
// alteração permite alterar o conteúdo das variáveis de memória

// Cria aHeader e aCols da GetDados
nUsado:=0
dbSelectArea("SX3")
dbSeek("SC6")

aHeader:={}

While !Eof().And.(x3_arquivo=="SC6")
    If Alltrim(x3_campo)=="C6_ITEM"
        dbSkip()
        Loop
    Endif
    If X3USO(x3_usado).And.cNivel>=x3_nivel
        nUsado:=nUsado+1
        Aadd(aHeader,{ TRIM(x3_titulo), x3_campo, x3_picture,;
                      x3_tamanho, x3_decimal,"AlwaysTrue()",;
                      x3_usado, x3_tipo, x3_arquivo, x3_context } )
    Endif
    dbSkip()
End

If nOpcx==3 // Incluir
```

```

aCols:={Array(nUsado+1)}
aCols[1,nUsado+1]:=F.
For nX:=1 to nUsado
    aCols[1,nX]:=CriaVar(aHeader[nX,2])
Next
Else
aCols:={}
dbSelectArea("SC6")
dbSetOrder(1)
dbSeek(xFilial()+M->C5_NUM)
While !eof().and.C6_NUM==M->C5_NUM
    AADD(aCols,Array(nUsado+1))
    For nX:=1 to nUsado

        aCols[Len(aCols),nX]:=FieldGet(FieldPos(aHeader[nX,2]))
        Next
        aCols[Len(aCols),nUsado+1]:=F.
        dbSkip()
    End
Endif
If Len(aCols)>0
    // Última linha da tabela
    //³ Executa a Modelo 3
3
    // Última linha da tabela
    aCposE := {"C5_CLIENTE"}

    IRetMod3 := Modelo3(cTitulo, cAliasE, cAliasG, aCposE, cLinOk,
cTudOk,;
                                nOpcE, nOpcG,cFieldOk)
    // Última linha da tabela
    //³ Executar processamento
3
    // Última linha da tabela
    If !RetMod3
        Aviso("Modelo3()", "Confirmada operacao!", {"Ok"})
    Endif
Endif
Return

```



### **Exercícios**

#### **Exercício 11**

Implementar uma Modelo3() para as tabelas padrões do ERP – SF1: Cab. NFE e SD1: Itens NFE

## 8. Arquivos e Índices Temporários

### 2.13 Utilização de arquivos e índices temporários

Os arquivos e índices temporários ou de trabalho, são geralmente utilizados em ambiente CodeBase, pois, neste ambiente não há os recursos de “Join” e “Order By”, como nos bancos de dados relacionais. Por este motivo, quando necessitar gerar uma informação ordenada e consolidada (ou seja, de várias tabelas), deveremos recorrer ao uso dos arquivos e dos índices temporários.

### 2.14 Funções para manipulação de arquivos e índices temporários

#### 2.14.1 CriaTrab()

A CriaTrab() é uma funcionalidade que permite criar um arquivo físico ou gerar um nome aleatório.

**Sintaxe:** `CriaTrab(aCampo, ICriar, cExt)`

##### Parâmetros

<b>aCampo</b>	Array com o nome, tipo, tamanho e decimal do campo a ser criado arquivo
<b>ICriar</b>	Se verdadeiro (.T.) criar o arquivo, ou falso (.F.) somente retorna um nome aleatório
<b>cExt</b>	Qual extensão deverá ser criado o arquivo de trabalho



*Importante*

Os arquivos criados com a função CRIATRAB() serão gerados no diretório especificado como “StartPath”, de acordo com o “RootPath” configurado no .ini da aplicação.

#### 2.14.2 dbUseArea()

A dbUseArea() é uma funcionalidade que permite definir um arquivo de base de dados, com uma área de trabalho disponível na aplicação.

**Sintaxe:** `dbUseArea(lNewArea, cDriver, cName, cAlias, lShared, lReadOnly)`

##### Parâmetros

<b>INewArea</b>	Indica se é um novo alias no conjunto de alias aberto.
<b>cDriver</b>	Drive (RddName()) do arquivo -> DBFCDX / TOPCONN / DBFNTX.
<b>cName</b>	Nome físico da tabela que será usado.
<b>cAlias</b>	Alias que será usado enquanto estiver aberto.
<b>IShared</b>	A tabela terá acesso exclusivo ou compartilhado.
<b>IReadOnly</b>	Se verdadeiro, a tabela será somente leitura.

## **2.14.3 IndRegua()**

A IndRegua() é uma funcionalidade que permite criar índices temporários para o alias especificado, podendo ou não ter um filtro.

**Sintaxe:** **IndRegua(cAlias, cNIndex, cExpress, xOrdem, cFor, cMens, IShow)**

### **Parâmetros**

<b>cAlias</b>	Alias da tabela onde será efetuada o índice/filtro temporário.
<b>cNIndex</b>	Nome do arquivo de trabalho retornado pela função CriaTrab().
<b>cExpress</b>	Expressão utilizada na chave do novo índice.
<b>xOrdem</b>	Parâmetro nulo.
<b>cFor</b>	Expressão utilizada para filtro.
<b>cMens</b>	Parâmetro nulo.
<b>IShow</b>	Apresentar a tela de progresso do índice/filtro temporário.

## **2.15 Funções Auxiliares para Arquivos de Trabalho e Temporários**

**DbStruct()** - Retorna um array com a estrutura de um Alias aberto no sistema:

Exemplo: {"campo","tipo",tamanho,decimal}

**RetIndex()** - Retorna a quantidade de índices ativa para um Alias aberto no sistema.

**DbSetIndex()** - Agrega um arquivo de índice a um Alias ativo no sistema.

Caso seja um índice temporário gerado pela IndRegua, somente deve ser agregado se Database principal não for Top.

**OrdBagExt()** - Retorna a extensão dos arquivos de índices utilizados pelo sistema (.idx / .cdx / .ntx)

### **Exemplo 01: Geração de arquivo e índice temporários**

```
#include "protheus.ch"

/*
+-----+
| Programa | GeraTrab | Autor | ROBSON LUIZ           | Data |
|          +-----+
| Desc.      | Utilização de arquivos e índices temporários
|          +-----+
| Uso        | Curso de ADVPL
|          +-----+
*/
User Function GeraTrab()

Local aStru
Local aArqTRB      := {}
Local nl           := 0
Local cIndTRB     := ""
Local cNomArq     := ""

AADD( aStru, { "PRODUTO"           , "B1_COD"           })
AADD( aStru, { "DESCRICAO"        , "B1_DESC"          })
AADD( aStru, { "GRUPO"             , "BM_GRUPO"        })
AADD( aStru, { "DESCGRUPO"         , "BM_DESC"          })
AADD( aStru, { "TIPO"              , "B1_TIPO"          })
```

```

AADD( aStru, { "DESCTIPO"      , "B1_DESC"           })
AADD( aStru, { "CC"            , "CTT_CC"           })
AADD( aStru, { "DESC_CC"       , "CTT_DESC"         })
AADD( aStru, { "SERIE"         , "D2_SERIE"         })
AADD( aStru, { "DOCTO"         , "D2_COD"          })
AADD( aStru, { "TIPONOTA"     , "D2_TP"           })
AADD( aStru, { "EMISSAO"      , "D2_EMISSAO"      })
AADD( aStru, { "CLIENTE"       , "D2_CLIENTE"      })
AADD( aStru, { "LOJA"          , "D2_LOJA"          })
AADD( aStru, { "NOME"          , "A1_NOME"         })
AADD( aStru, { "QTDE"          , "D2_QUANT"        })
AADD( aStru, { "UNIT"          , "D2_PRCVEN"      })
AADD( aStru, { "TOTAL"         , "D2_TOTAL"        })
AADD( aStru, { "ALIQICMS"     , "D2_PICM"         })
AADD( aStru, { "VALICMS"       , "D2_VALICM"       })
AADD( aStru, { "ALIQIPI"       , "D2_IPI"          })
AADD( aStru, { "VALIPI"        , "D2_VALIPI"       })
AADD( aStru, { "VALMERC"       , "D2_TOTAL"        })
AADD( aStru, { "TOTSEMICMS"   , "D2_TOTAL"        })
AADD( aStru, { "VALPIS"         , "D2_TOTAL"        })
AADD( aStru, { "LIQUIDO"       , "D2_TOTAL"        })
AADD( aStru, { "CUSTO"         , "D2_TOTAL"        })

dbSelectArea("SX3")
dbSetOrder(2)
For nl := 1 To Len( aStru )
    dbSeek( aStru[nl,2] )
    AADD( aArqTRB, { aStru[nl,1], X3_TIPO, X3_TAMANHO, X3_DECIMAL } )
)
Next nl

// Índice que será criado
cIndTRB := "PRODUTO+DTOS(EMISSAO)"

cNomArq := CriaTrab( aArqTRB, .T. )

dbUseArea( .T., "DBFCDX", cNomArq, "TRB", .T. ,.T. )

IndRegua( "TRB", cNomArq, cIndTRB )
dbSetOrder(1)

// ( ... ) fazer o processamento necessário

dbSelectArea("TRB")
dbCloseArea()

If MsgYesNo("Apagar o arquivo gerado \system\"+cNomArq+".dbf
?",FunName())
    Ferase(cNomArq+".dbf")

```

```

Ferase(cNomArq+OrdBagExt())
Endif

Return Nil

```



**Importante**

- Quando criamos um arquivo ou um índice temporário (trabalho), utilizando a função *Indregua*, é obrigatório apagá-los no final do rotina.
- A utilização de arquivo ou índice temporário, deverá ser bem analisada a fim de evitar lentidão nos processamentos da rotina.



**Dica**

- O array **aStru** foi criado com base nos campos existentes no sistema, ao invés de criarmos novas estruturas dos campos, utilizamos as já existentes no Dicionários de Dados (SX3).

## Exemplo 02: Utilizando dois índices temporários com RDD DBFCDX

```

/*
+-----
-----+
| Programa | IndTwoReg | Autor | MICHEL DANTAS           | Data |
|          +-----+
+-----+
| Desc.      | Utilização de dois índices temporários com DBFCDX
|          +-----+
-----+
| Uso        | Curso de ADVPL
|          +-----+
+-----+
*/
User Function IndTwoReg()

LOCAL nOrder := 0
LOCAL cArq1 := CriaTrab(NIL,.F.)
LOCAL cChave1 := "A1_FILIAL+A1_EST"
LOCAL cArq2 := CriaTrab(NIL,.F.)
LOCAL cChave2 := "A1_FILIAL+A1_NOME"

dbSelectArea("SA1")
IndRegua("SA1",cArq1,cChave1,,, "Selecionando Regs... ")
IndRegua("SA1",cArq2,cChave2,,, "Selecionando Regs... ")

```

```

nOrder := RetIndex("SA1")

dbSetIndex(cArq1+OrdBagExt())
dbSetIndex(cArq2+OrdBagExt())

Alert("Agora vai por estado")

dbsetOrder(nOrder+1)
dbGoTop()
While !Eof()
    Alert("Estado : " + SA1->A1_EST +"+" Nome : " + SA1-
>A1_NOME)
    dbSkip()
End

Alert("Agora vai por nome")

dbSetOrder(nOrder+2)
dbGoTop()
While !Eof()
    Alert("Estado : " + SA1->A1_EST +"+" Nome : " + SA1-
>A1_NOME)
    dbSkip()
End

RetIndex("SA1")
Ferase(cArq1+OrdBagext())
Ferase(cArq2+OrdBagext())

Return

```



### **Exercícios**

#### **Exercício 12**

Implementar uma rotina que crie um arquivo de trabalho com uma estrutura similar ao SA1, e que receba as informações desta tabela.

#### **Exercício 13**

Implementar uma rotina que crie um arquivo de trabalho com uma estrutura similar ao SB1, e que receba as informações desta tabela.

## **9. Relatórios não gráficos**

Os relatórios desenvolvidos em ADVPL possuem um padrão de desenvolvimento que mais depende de layout e tipos de parâmetros do que qualquer outro tipo de informação, visto que até o momento percebemos que a linguagem padrão da Microsiga é muito mais composta de funções genéricas do que de comandos.

Este tipo de relatório é caracterizado por um formato de impressão tipo PostScript®, e permite a geração de um arquivo em formato texto (.txt), com uma extensão própria da aplicação ERP (.##R).

A estrutura de um relatório não gráfico é baseada no uso da função SetPrint(), complementada pelo uso de outras funções acessórias, as quais estão detalhadas no Guia de Referência Rápida que acompanha este material.

### **2.16 Funções Utilizadas para Desenvolvimento de Relatórios**

#### **2.16.1 SetPrint()**

A função que cria a interface com as opções de configuração para impressão de um relatório no formato texto. Basicamente duas variáveis **m\_pag** e **aReturn** precisam ser declaradas como privadas (private) antes de executar a SetPrint(). Após confirmada, os dados são armazenados no vetor aReturn que será passado como parâmetro para função SetDefault().

**Sintaxe:**   **SetPrint ( < cAlias > , < cProgram > , [ cPergunte ] , [ cTitle ] , [ cDesc1 ] , [ cDesc2 ] , [ cDesc3 ] , [ IDic ] , [ aOrd ] , [ ICompres ] , [ cSize ] , [ uParm12 ] , [ IFilter ] , [ ICrystal ] , [ cNameDrv ] , [ uParm16 ] , [ IServer ] , [ cPortPrint ] ) --> cReturn**

#### **Parâmetros:**

<b>cAlias</b>	Alias do arquivo a ser impresso.
<b>cProgram</b>	Nome do arquivo a ser gerado em disco.
<b>cPergunte</b>	Grupo de perguntas cadastrado no dicionário SX1.
<b>cTitle</b>	Título do relatório.
<b>cDesc1</b>	Descrição do relatório.
<b>cDesc2</b>	Continuação da descrição do relatório.
<b>cDesc3</b>	Continuação da descrição do relatório.
<b>IDic</b>	Utilizado na impressão de cadastro genérico permite a escolha dos campos a serem impressos. Se o parametro cAlias não for informado o valor padrão assumido será .F..
<b>aOrd</b>	Ordem(s) de impressão.
<b>ICompres</b>	Se verdadeiro (.T.) permite escolher o formato da impressão, o valor padrão

	assumido será .T.
<b>cSize</b>	Tamanho do relatório "P","M" ou "G".
<b>uParm12</b>	Parâmetro reservado.
<b>lFilter</b>	Se verdadeiro (.T.) permite a utilização do assistente de filtro, o valor padrão assumido será .T.
<b>ICrystal</b>	Se verdadeiro (.T.) permite integração com Crystal Report, o valor padrão assumido será .F.
<b>cNameDrv</b>	Nome de um driver de impressão.
<b>uParm16</b>	Parâmetro reservado.
<b>lServer</b>	Se verdadeiro (.T.) força impressão no servidor.
<b>cPortPrint</b>	Define uma porta de impressão padrão.

**Retorno:**

<b>Caracter</b>	Nome do Relatório
-----------------	-------------------

**Estrutura aReturn:**

<b>aReturn[1]</b>	Caracter, tipo do formulário.
<b>aReturn[2]</b>	Numérico, opção de margem.
<b>aReturn[3]</b>	Caracter, destinatário.
<b>aReturn[4]</b>	Numérico, formato da impressão.
<b>aReturn[5]</b>	Numérico, dispositivo de impressão.
<b>aReturn[6]</b>	Caracter, driver do dispositivo de impressão.
<b>aReturn[7]</b>	Caracter, filtro definido pelo usuário.
<b>aReturn[8]</b>	Numérico, ordem.
<b>aReturn[x]</b>	A partir a posição [9] devem ser informados os nomes dos campos que devem ser considerados no processamento, definidos pelo uso da opção Dicionário da SetPrint().

## 2.16.2 SetDefault()

A função SetDefault() prepara o ambiente de impressão de acordo com as informações configuradas no array aReturn, obtidas através da função SetPrint().

**Sintaxe:** **SetDefault ( < aReturn > , < cAlias > , [ uParm3 ] , [ uParm4 ] , [cSize] , [ nFormat ] )**

**Parâmetros:**

<b>aReturn</b>	Configurações de impressão.
<b>cAlias</b>	Alias do arquivo a ser impresso.
<b>uParm3</b>	Parâmetro reservado.
<b>uParm4</b>	Parâmetro reservado.
<b>cSize</b>	Tamanho da página "P","M" ou "G"
<b>nFormat</b>	Formato da página, 1 retrato e 2 paisagem.

**Estrutura aReturn:**

<b>aReturn[1]</b>	Caracter, tipo do formulário.
<b>aReturn[2]</b>	Numérico, opção de margem.
<b>aReturn[3]</b>	Caracter, destinatário.
<b>aReturn[4]</b>	Numérico, formato da impressão.
<b>aReturn[5]</b>	Numérico, dispositivo de impressão.
<b>aReturn[6]</b>	Caracter, driver do dispositivo de impressão.
<b>aReturn[7]</b>	Caracter, filtro definido pelo usuário.
<b>aReturn[8]</b>	Numérico, ordem.
<b>aReturn[x]</b>	A partir a posição [9] devem ser informados os nomes dos campos que devem ser considerados no processamento, definidos pelo uso da opção Dicionário da SetPrint().

**2.16.3 RptStatus()**

Régua de processamento simples, com apenas um indicador de progresso, utilizada no processamento de relatórios do padrão SetPrint().

**Sintaxe: RptStatus(bAcao, cMensagem)**

**Parâmetros:**

<b>bAcao</b>	Bloco de código que especifica a ação que será executada com o acompanhamento da régua de processamento.
<b>cMensagem</b>	Mensagem que será exibida na régua de processamento durante a execução.

### **2.16.3.1 SETREGUA()**

---

A função SetRegua() é utilizada para definir o valor máximo da régua de progressão criada através da função RptStatus().

**Sintaxe:** **SetRegua(nMaxProc)**

**Parâmetros:**

<b>nMaxProc</b>	Variável que indica o valor máximo de processamento (passos) que serão indicados pela régua.
-----------------	--

### **2.16.3.2 INCREGUA()**

---

A função IncRegua() é utilizada para incrementar valor na régua de progressão criada através da função RptStatus()

**Sintaxe:** **IncRegua(cMensagem)**

**Parâmetros:**

<b>cMensagem</b>	Mensagem que será exibida e atualizada na régua de processamento a cada execução da função IncRegua(), sendo que a taxa de atualização da interface é controlada pelo Binário.
------------------	--

## **2.16.4 CABEC()**

A função CABEC() determina as configurações de impressão do relatório e imprime o cabeçalho do mesmo.

**Sintaxe:** **Cabec(cTitulo, cCabec1, cCabec2, cNomeProg, nTamanho, nCompress, aCustomText, IPerg, cLogo)**

**Parâmetros:**

<b>cTitulo</b>	Título do relatório.
<b>cCabec1</b>	String contendo as informações da primeira linha do cabeçalho.
<b>cCabec2</b>	String contendo as informações da segunda linha do cabeçalho.
<b>cNomeProg</b>	Nome do programa de impressão do relatório.
<b>nTamanho</b>	Tamanho do relatório em colunas (80, 132 ou 220).
<b>nCompress</b>	Indica se impressão será comprimida (15) ou normal (18).
<b>aCustomText</b>	Texto específico para o cabeçalho, substituindo a estrutura padrão do sistema.
<b>IPerg</b>	Permite a supressão da impressão das perguntas do relatório, mesmo que o parâmetro MV_IMPSX1 esteja definido como "S".

#### **Parâmetros (continuação):**

<b>cLogo</b>	Redefine o bitmap que será impresso no relatório, não necessitando que ele esteja no formato padrão da Microsiga: "LGRL"+SM0->M0_CODIGO+SM0->M0_CODFIL+".BMP"
--------------	--

#### **2.16.5 RODA()**

A função RODA() imprime o rodapé da página do relatório, o que pode ser feito a cada página, ou somente ao final da impressão.

**Sintaxe:** Roda(**uPar01, uPar02, cSize**)

#### **Parâmetros:**

<b>uPar01</b>	Não é mais utilizado.
<b>uPar02</b>	Não é mais utilizado.
<b>cSize</b>	Tamanho do relatório ("P","M","G").

#### **2.16.6 Pergunte()**

A função PERGUNTE() inicializa as variáveis de pergunta (mv\_par01,...) baseada na pergunta cadastrado no Dicionário de Dados (SX1). Se o parâmetro IAsk não for especificado, ou for verdadeiro, será exibida a tela para edição da pergunta, e, se o usuário confirmar, as variáveis serão atualizadas e a pergunta no SX1 também será atualizada.

**Sintaxe:** Pergunte( **cPergunta , [IAsk] , [cTitle]** )

#### **Parâmetros:**

<b>cPergunta</b>	Pergunta cadastrada no dicionário de dados ( SX1 ) a ser utilizada.
<b>/Ask</b>	Indica se exibirá a tela para edição.
<b>cTitle</b>	Título do diálogo.

#### **Retorno:**

<b>Lógico</b>	Indica se a tela de visualização das perguntas foi confirmada (.T.) ou cancelada (.F.)
---------------	--

#### **2.16.7 AjustaSX1()**

A função AJUSTASX1() permite a inclusão simultânea de vários itens de perguntas para um grupo de perguntas no SX1 da empresa ativa.

## **Sintaxe: AJUSTASX1(cPerg, aPergs)**

**Parâmetros:**

<b>cPerg</b>	Grupo de perguntas do SX1 (X1_GRUPO)
<b>aPergs</b>	Array contendo a estrutura dos campos que serão gravados no SX1.

**Estrutura – Item do array aPerg:**

<b>Posição</b>	<b>Campo</b>	<b>Tipo</b>	<b>Descrição</b>
<b>01</b>	X1_PERGUNT	Caractere	Descrição da pergunta em português.
<b>02</b>	X1_PERSPA	Caractere	Descrição da pergunta em espanhol.
<b>03</b>	X1_PERENG	Caractere	Descrição da pergunta em inglês.
<b>04</b>	X1_VARIAVL	Caractere	Nome da variável de controle auxiliar (mv_ch).
<b>05</b>	X1_TIPO	Caractere	Tipo do parâmetro.
<b>06</b>	X1_TAMANHO	Numérico	Tamanho do conteúdo do parâmetro.
<b>07</b>	X1_DECIMAL	Numérico	Número de decimais para conteúdos numéricos.
<b>08</b>	X1_PRESEL	Numérico	Define qual opção do combo é o padrão para o parâmetro.
<b>09</b>	X1_GSC	Caractere	Define se a pergunta será do tipo G – Get ou C – Choice (combo).
<b>10</b>	X1_VALID	Caractere	Expressão de validação do parâmetro.
<b>11</b>	X1_VAR01	Caractere	Nome da variável MV_PAR+”Ordem” do parâmetro.
<b>12</b>	X1_DEF01	Caractere	Descrição da opção 1 do combo em português.
<b>13</b>	X1_DEFSPA1	Caractere	Descrição da opção 1 do combo em espanhol.
<b>14</b>	X1_DEFENG1	Caractere	Descrição da opção 1 do combo em inglês.
<b>15</b>	X1_CNT01	Caractere	Conteúdo padrão ou último conteúdo definido como respostas para a pergunta.
<b>16</b>	X1_VAR02	Caractere	Não é informado.
<b>17</b>	X1_DEF02	Caractere	Descrição da opção X do combo em português.
<b>18</b>	X1_DEFSPA2	Caractere	Descrição da opção X do combo em espanhol.
<b>19</b>	X1_DEFENG2	Caractere	Descrição da opção X do combo em inglês.
<b>20</b>	X1_CNT02	Caractere	Não é informado.
<b>21</b>	X1_VAR03	Caractere	Não é informado.
<b>22</b>	X1_DEF03	Caractere	Descrição da opção X do combo em

			português.
<b>23</b>	X1_DEFSPA3	Caractere	Descrição da opção X do combo em espanhol.
<b>24</b>	X1_DEFENG3	Caractere	Descrição da opção X do combo em inglês.
<b>25</b>	X1_CNT03	Caractere	Não é informado.
<b>26</b>	X1_VAR04	Caractere	Não é informado.
<b>27</b>	X1_DEF04	Caractere	Descrição da opção X do combo em português.
<b>28</b>	X1_DEFSPA4	Caractere	Descrição da opção X do combo em espanhol.
<b>29</b>	X1_DEFENG4	Caractere	Descrição da opção X do combo em inglês.
<b>30</b>	X1_CNT04	Caractere	Não é informado.
<b>31</b>	X1_VAR05	Caractere	Não é informado.
<b>32</b>	X1_DEF05	Caractere	Descrição da opção X do combo em português.
<b>33</b>	X1_DEFSPA5	Caractere	Descrição da opção X do combo em espanhol.
<b>34</b>	X1_DEFENG5	Caractere	Descrição da opção X do combo em inglês.
<b>35</b>	X1_CNT05	Caractere	Não é informado.
<b>36</b>	X1_F3	Caractere	Código da consulta F3 vinculada ao parâmetro.
<b>37</b>	X1_GRPSXG	Caractere	Código do grupo de campos SXG para atualização automática, quando o grupo for alterado.

<b>38</b>	X1_PYME	Caractere	Se a pergunta estará disponível no ambiente Pyme.
<b>39</b>	X1_HELP	Caractere	Conteúdo do campo X1_HELP.
<b>40</b>	X1_PICTURE	Caractere	Picture de formatação do conteúdo do campo.
<b>41</b>	aHelpPor	Array	Vetor simples contendo as linhas de help em português para o parâmetro. Trabalhar com linhas de até 40 caracteres.
<b>42</b>	aHelpEng	Array	Vetor simples contendo as linhas de help em inglês para o parâmetro. Trabalhar com linhas de até 40 caracteres.
<b>43</b>	aHelpSpa	Array	Vetor simples contendo as linhas de help em espanhol para o parâmetro. Trabalhar com linhas de até 40 caracteres.

## 2.16.8PutSX1()

A função PUTSX1() permite a inclusão de um único item de pergunta em um grupo definido no Dicionário de Dados (SX1). Todos os vetores contendo os textos explicativos da pergunta devem conter até 40 caracteres por linha.

**Sintaxe:** PutSx1(cGrupo, cOrdem, cPergunt, cPerSpa, cPerEng, cVar, cTipo,  
 nTamanho, nDecimal, nPresel, cGSC, cValid, cF3, cGrpSxg  
 ,cPyme, cVar01, cDef01, cDefSpa1 , cDefEng1, cCnt01, cDef02,  
 cDefSpa2, cDefEng2, cDef03, cDefSpa3, cDefEng3, cDef04,  
 cDefSpa4, cDefEng4, cDef05, cDefSpa5, cDefEng5, aHelpPor,  
 aHelpEng, aHelpSpa, cHelp)

**Parâmetros:**

<b>cGrupo</b>	Grupo de perguntas do SX1 (X1_GRUPO).
<b>cOrdem</b>	Ordem do parâmetro no grupo (X1_ORDEM).
<b>cPergunt</b>	Descrição da pergunta em português.
<b>cPerSpa</b>	Descrição da pergunta em espanhol.
<b>cPerEng</b>	Descrição da pergunta em inglês.
<b>cVar</b>	Nome da variável de controle auxiliar (X1_VARIAVL).
<b>cTipo</b>	Tipo do parâmetro.
<b>nTamanho</b>	Tamanho do conteúdo do parâmetro.
<b>nDecimal</b>	Número de decimais para conteúdos numéricos.
<b>nPresel</b>	Define qual opção do combo é o padrão para o parâmetro.
<b>cGSC</b>	Define se a pergunta será do tipo G – Get ou C – Choice (combo).
<b>cValid</b>	Expressão de validação do parâmetro.
<b>cF3</b>	Código da consulta F3 vinculada ao parâmetro
<b>cGrpSxg</b>	Código do grupo de campos SXG para atualização automática, quando o grupo for alterado.
<b>cPyme</b>	Se a pergunta estará disponível no ambiente Pyme.
<b>cVar01</b>	Nome da variável MV_PAR+”Ordem” do parâmetro.
<b>cDef01</b>	Descrição da opção 1 do combo em português.
<b>cDefSpa1</b>	Descrição da opção 1 do combo em espanhol.
<b>cDefEng1</b>	Descrição da opção 1 do combo em inglês.
<b>cCnt01</b>	Conteúdo padrão ou último conteúdo definido como respostas para este item.
<b>cDef0x</b>	Descrição da opção X do combo em português.
<b>cDefSpax</b>	Descrição da opção X do combo em espanhol.
<b>cDefEngx</b>	Descrição da opção X do combo em inglês.
<b>aHelpPor</b>	Vetor simples contendo as linhas de help em português para o parâmetro.
<b>aHelpEng</b>	Vetor simples contendo as linhas de help em inglês para o parâmetro.
<b>aHelpSpa</b>	Vetor simples contendo as linhas de help em espanhol para o parâmetro.
<b>cHelp</b>	Conteúdo do campo X1_HELP.

## **2.17 Estrutura de Relatórios Baseados na SetPrint()**

Neste tópico será demonstrada a construção de relatório não gráfico baseado no uso da função SetPrint() o qual atende os formatos de base de dados ISAM e Top Connect. Porém, não contemplando a tecnologia Protheus Embedded SQL.

### **Estrutura do programa**

---

<b>Linhas Programa</b>	
1	<b>Função principal;</b>
2	Declaração e atribuição de variáveis;
3	Atualização do arquivo de perguntas através da função específica
	CriaSX1();
4	Definição das perguntas através da função Pergunte();
5	Definição das ordens disponíveis para impressão do relatório;
6	Chamada da função SetPrint;
7	Atualização das configurações de impressão com a função SetDefault();
8	Execução da rotina de impressão através da função RptStatus()
9	<b>Fim da função principal.</b>
10	<b>Função de processamento e impressão do relatório</b>
11	Declaração e atribuição de variáveis;
12	Definição dos filtros de impressão, avaliando o banco de dados em uso
	pela aplicação;
13	Atualização da régua de processamento com a quantidade de registros
	que será processada;
14	Estrutura principal de repetição para impressão dos dados do relatório;
15	Controle da impressão do cabeçalho utilizando a função
	Cabec();
16	Impressão dos totais do relatório;
17	Impressão do rodapé da última página do relatório utilizando a função
	Roda();
18	Limpeza dos arquivos e índices temporários criados para o
	processamento();
19	Tratamento da visualização do relatório (impressão em disco) através
	da função OurSpool()
20	Tratamentos adicionais ao relatório, de acordo com necessidades
	específicas;
21	Liberação do buffer de impressão, seja para impressora, seja para
	limpeza do conteúdo
	visualizado em tela, utilizando a função MS_FLUSH()
22	<b>Fim da função de processamento e impressão do relatório</b>
23	<b>Função de atualização do arquivo de perguntas</b>
24	Declaração e atribuição de variáveis;
25	Opção 01: Adição individual de cada pergunta no SX1 utilizando a

- função PUTSX1()  
 Criação de um array individual no formato utilizado pela  
 PUSTX1() contendo apenas as informações da pergunta que será  
 adicionada no SX1.
- 25      Opção 02: Adição de um grupo de perguntas no SX1 utilizando a  
 função AJUSTASX1()  
 Criação de um array no formato utilizado pela AJUSTASX1()  
 contendo todas as perguntas que serão atualizadas.

**26      Fim da função de atualização do arquivo de perguntas**

---

### Função Principal

```
//+-----
+
//| Rotina | Inform | Autor | Robson Luiz (rleg) | Data | 01.01.07 |
//+-----
+
//| Descr. | Rotina para gerar relatório utilizando as funções
|
//|           | SetPrint() e SetDefault().
|
//+-----
+
//| Uso      | Para treinamento e capacitação.
|
//+-----
+
User Function INFORM()
//+-----
//| Declarações de variáveis
//+-----

Local cDesc1  := "Este relatório irá imprimir informações do contas
a pagar conforme"
Local cDesc2  := "parâmetros informado. Será gerado um arquivo no
diretório "
Local cDesc3  := "Spool - INFORM_????.XLS, onde ???? é o nome do
usuário."

Private cString  := "SE2"
Private Tamanho  := "M"
Private aReturn  := { "Zebrado",2,"Administração",2,2,1,"",1 }
Private wnrrel   := "INFORM"
Private NomeProg := "INFORM"
Private nLastKey := 0
Private Limite   := 132
```

```

Private Titulo := "Título a Pagar - Ordem de "
Private cPerg := "INFORM"
Private nTipo := 0
Private cbCont := 0
Private cbTxt := "registro(s) lido(s)"
Private Li := 80
Private m_pag := 1
Private aOrd := {}
Private Cabec1 := "PREFIXO TITULO PARCELA TIP EMISSAO VENCTO
VENCTO"
Private Cabec1 += "REAL VLR. ORIGINAL          PAGO
SALDO "
Private Cabec2 := ""
/*
+-----
| Parâmetros do aReturn
+-----
aReturn - Preenchido pelo SetPrint()
aReturn[1] - Reservado para formulário
aReturn[2] - Reservado para numero de vias
aReturn[3] - Destinatário
aReturn[4] - Formato 1=Paisagem 2=Retrato
aReturn[5] - Mídia 1-Disco 2=Impressora
aReturn[6] - Porta ou arquivo 1-Lpt1... 4-Com1...
aReturn[7] - Expressão do filtro
aReturn[8] - Ordem a ser selecionada
aReturn[9] [10] [n] - Campos a processar se houver
*/
AADD( aOrd, "Fornecedor" )
AADD( aOrd, "Titulo" )
AADD( aOrd, "Emissão" )
AADD( aOrd, "Vencimento" )
AADD( aOrd, "Vencto. Real" )

//Parâmetros de perguntas para o relatório
//+-----+
//| mv_par01 - Fornecedor de      ? 999999 |
//| mv_par02 - Fornecedor ate    ? 999999 |
//| mv_par03 - Tipo de          ? XXX   |
//| mv_par04 - Tipo ate         ? XXX   |
//| mv_par05 - Vencimento de    ? 99/99/99 |
//| mv_par06 - Vencimento ate   ? 99/99/99 |
//| mv_par07 - Aglut.Fornecedor ? Sim/Não |
//+-----+
CriaSx1()

//+-----
//| Disponibiliza para usuário digitar os parâmetros

```

```

//+-----
Pergunte(cPerg,.F.)
//cPerg -> Nome do grupo de perguntas, .T. mostra a tela,;
// .F. somente carrega as variáveis

//+-----
//| Solicita ao usuário a parametrização do relatório.
//+-----

wnrel :=
SetPrint(cString,wnrel,cPerg,@Titulo,cDesc1,cDesc2,cDesc3,.F.,aOrd,.
F., ;
Tamanho,.F.,.F.)
//SetPrint(cAlias,cNome,cPerg,cDesc,cCnt1,cCnt2,cCnt3,LDic,aOrd,lCom
pres,;
//cSize,aFilter,lFiltro,lCrystal,cNameDrv,lNoAsk,lServer,cPortToPrin
t)

//+-----
//| Se teclar ESC, sair
//+-----

If nLastKey == 27
Return
Endif

//+-----
-
//| Estabelece os padrões para impressão, conforme escolha do
usuário
//+-----
-
SetDefault(aReturn,cString)

//+-----
//| Verificar se será reduzido ou normal
//+-----

nTipo := IIF(aReturn[4] == 1, 15, 18)

//+-----
//| Se teclar ESC, sair
//+-----

If nLastKey == 27
Return
Endif

//+-----
//| Chama função que processa os dados
//+-----

RptStatus({|lEnd| ImpRel(@lEnd) }, Titulo, "Processando e imprimindo
dados,;
aguarde...", .T. )

```

Return

## Função de processamento e impressão

```
//+-----  
+  
//| Rotina | ImpRel | Autor | Robson Luiz (rleg) | Data | 01.01.07 |  
//+-----  
+  
//| Descr. | Rotina de processamento e impressão.  
|  
//+-----  
+  
//| Uso      | Para treinamento e capacitação.  
|  
//+-----  
+  
  
Static Function ImpRel(lEnd)  
  
Local nIndice := 0  
Local cArq := ""  
Local cIndice := ""  
Local cFiltro := ""  
Local aCol := {}  
Local cFornec := ""  
Local nValor := 0  
Local nPago := 0  
Local nSaldo := 0  
Local nT_Valor := 0  
Local nT_Pago := 0  
Local nT_Saldo := 0  
Local cArqExcel := ""  
Local cAliasImp  
Local oExcelApp  
  
Titulo += aOrd[aReturn[8]]  
  
#IFNDEF TOP  
cAliasImp := "SE2"  
  
cFiltro := "E2_FILIAL == ""+xFilial("SE2")+"" "  
cFiltro += ".And. E2_FORNECE >= ""+mv_par01+"" "  
cFiltro += ".And. E2_FORNECE <= ""+mv_par02+"" "  
cFiltro += ".And. E2_TIPO >= ""+mv_par03+"" "  
cFiltro += ".And. E2_TIPO <= ""+mv_par04+"" "  
cFiltro += ".And. Dtos(E2_VENCTO) >= ""+Dtos(mv_par05)+"" "
```

```

cFiltro += ".And. Dtos(E2_VENCTO) <= "+Dtos(mv_par06)+" "
If aReturn[8] == 1 //Fornecedor
    cIndice := "E2_FORNECE+E2_LOJA+E2_NUM"
Elseif aReturn[8] == 2 //Titulo
    cIndice := "E2_NUM+E2_FORNECE+E2_LOJA"
Elseif aReturn[8] == 3 //Emissao
    cIndice := "Dtos(E2_EMISSAO)+E2_FORNECE+E2_LOJA"
Elseif aReturn[8] == 4 //Vencimento
    cIndice := "Dtos(E2_VENCTO)+E2_FORNECE+E2_LOJA"
Elseif aReturn[8] == 5 //Vencimento Real
    cIndice := "Dtos(E2_VENCREA)+E2_FORNECE+E2_LOJA"
Endif

cArq := CriaTrab(NIL,,F.)
dbSelectArea(cAliasImp)
IndRegua(cAliasImp,cArq,cIndice,,cFiltro)
nIndice := RetIndex()
nIndice := nIndice + 1
dbSetIndex(cArq+OrdBagExt())
dbSetOrder(nIndice)
#ELSE
cAliasImp := GetNextAlias()

cQuery := "SELECT "
cQuery += "E2_PREFIJO, E2_NUM, E2_PARCELA, E2_TIPO, E2_FORNECE,
E2_LOJA, E2_NOMFOR, "
cQuery += "E2_EMISSAO, E2_VENCTO, E2_VENCREA, E2_VALOR, E2_SALDO "
cQuery += "FROM "+RetSqlName("SE2")+" "
cQuery += "WHERE E2_FILIAL = "+xFilial("SE2")+" "
cQuery += "AND E2_FORNECE >= "+mv_par01+" "
cQuery += "AND E2_FORNECE <= "+mv_par02+" "
cQuery += "AND E2_TIPO >= "+mv_par03+" "
cQuery += "AND E2_TIPO <= "+mv_par04+" "
cQuery += "AND E2_VENCTO >= "+Dtos(mv_par05)+" "
cQuery += "AND E2_VENCTO <= "+Dtos(mv_par06)+" "
cQuery += "AND D_E_L_E_T_ <> '*' "
cQuery += "ORDER BY "

If aReturn[8] == 1 //Fornecedor
    cQuery += "E2_FORNECE,E2_LOJA,E2_NUM"
Elseif aReturn[8] == 2 //Titulo
    cQuery += "E2_NUM,E2_FORNECE,E2_LOJA"
Elseif aReturn[8] == 3 //Emissao
    cQuery += "E2_EMISSAO,E2_FORNECE,E2_LOJA"
Elseif aReturn[8] == 4 //Vencimento
    cQuery += "E2_VENCTO,E2_FORNECE,E2_LOJA"
Elseif aReturn[8] == 5 //Vencimento Real

```

```

        cQuery += "E2_VENCREA,E2_FORNECE,E2_LOJA"
Endif

dbUseArea( .T., "TOPCONN", TcGenQry(,,cQuery), cAliasImp, .T., .F. )
dbSelectArea(cAliasImp)

/* Instrução SQL Embedded
-----
If aReturn[8] == 1 //Fornecedor
    cOrder := "E2_FORNECE,E2_LOJA,E2_NUM"
Elseif aReturn[8] == 2 //Título
    cOrder := "E2_NUM,E2_FORNECE,E2_LOJA"
Elseif aReturn[8] == 3 //Emissão
    cOrder := "E2_EMISSAO,E2_FORNECE,E2_LOJA"
Elseif aReturn[8] == 4 //Vencimento
    cOrder := "E2_VENCTO,E2_FORNECE,E2_LOJA"
Elseif aReturn[8] == 5 //Vencimento Real
    cOrder := "E2_VENCREA,E2_FORNECE,E2_LOJA"
Endif

BeginSQL Alias cAliasImp
    Column E2_EMISSAO As Date
    Column E2_VENCTO As Date
    Column E2_VENCREA As Date
    Column E2_VALOR      As Numeric(12)
    Column E2_SALDO      As Numeric(12)
    %NoParser%

        SELECT      E2_PREFIXO, E2_NUM, E2_PARCELA, E2_TIPO,
E2_FORNECE,
                E2_LOJA, E2_NOMFOR, E2_EMISSAO, E2_VENCTO, E2_VENCREA,
E2_VALOR,
                E2_SALDO
        FROM       %Table:SE2
        WHERE
                E2_FILIAL = %xFilial% AND
                E2_FORNECE BETWEEN %Exp:mv_par01% AND %Exp:mv_par02% AND
                E2_TIPO BETWEEN %Exp:mv_par03% AND %Exp:mv_par04% AND
                E2_VENCTO BETWEEN %Exp:mv_par05% AND %Exp:mv_par06% AND
                %NotDel%
        ORDER BY %Order:cOrder%
EndSQL
*/
#endif

dbGoTop()
SetRegua(0)

```

```

//+-----
//| Coluna de impressão
//+-----
AADD( aCol, 004 ) //Prefixo
AADD( aCol, 012 ) //Titulo
AADD( aCol, 024 ) //Parcela
AADD( aCol, 031 ) //Tipo
AADD( aCol, 036 ) //Emissao
AADD( aCol, 046 ) //Vencimento
AADD( aCol, 058 ) //Vencimento Real
AADD( aCol, 070 ) //Valor Original
AADD( aCol, 090 ) //Pago
AADD( aCol, 110 ) //Saldo

cFornec := (cAliasImp)->E2_FORNECE+(cAliasImp)->E2_LOJA

While !Eof() .And. !End

If Li > 55
    Cabec(Titulo,Cabec1,Cabec2,NomeProg,Tamanho,nTipo)
Endif

@ Li, aCol[1] PSay "Cod/Loj/Nome: "+(cAliasImp)->E2_FORNECE+;
"-"+(cAliasImp)->E2_LOJA+" "+(cAliasImp)->E2_NOMFOR
Li ++

While !Eof() .And. !End .And. ;
(cAliasImp)->E2_FORNECE+(cAliasImp)->E2_LOJA == cFornec

IncRegua()

If Li > 55
    Cabec(Titulo,Cabec1,Cabec2,NomeProg,Tamanho,nTipo)
Endif

If mv_par07 == 2
@ Li, aCol[1] PSay (cAliasImp)->E2_PREFIXO
    @ Li, aCol[2] PSay (cAliasImp)->E2_NUM
    @ Li, aCol[3] PSay (cAliasImp)->E2_PARCELA
    @ Li, aCol[4] PSay (cAliasImp)->E2_TIPO
    @ Li, aCol[5] PSay (cAliasImp)->E2_EMISSAO
    @ Li, aCol[6] PSay (cAliasImp)->E2_VENCTO
    @ Li, aCol[7] PSay (cAliasImp)->E2_VENCREA
    @ Li, aCol[8] PSay (cAliasImp)->E2_VALOR ;
    PICTURE "@E 99,999,999,999.99"
    @ Li, aCol[9] PSay (cAliasImp)->E2_VALOR -;
(cAliasImp)->E2_SALDO ;
    PICTURE "@E 99,999,999,999.99"

```

```

        @ Li, aCol[10] PSay (cAliasImp)->E2_SALDO ;
        PICTURE "@E 99,999,999,999.99"
        Li ++
    Endif

    nValor += (cAliasImp)->E2_VALOR
    nPago   += ((cAliasImp)->E2_VALOR-(cAliasImp)->E2_SALDO)
    nSaldo += (cAliasImp)->E2_SALDO

    nT_Valor += (cAliasImp)->E2_VALOR
    nT_Pago  += ((cAliasImp)->E2_VALOR-(cAliasImp)->E2_SALDO)
    nT_Saldo += (cAliasImp)->E2_SALDO

    dbSkip()

End

@ Li, 000 PSay Replicate("-",Limite)
Li ++
@ Li, aCol[1]  PSay "TOTAL....."
@ Li, aCol[8]  PSay nValor PICTURE "@E 99,999,999,999.99"
@ Li, aCol[9]  PSay nPago   PICTURE "@E 99,999,999,999.99"
@ Li, aCol[10] PSay nSaldo PICTURE "@E 99,999,999,999.99"
Li +=2

cFornec := (cAliasImp)->E2_FORNECE+(cAliasImp)->E2_LOJA
nValor  := 0
nPago   := 0
nSaldo  := 0

End

If lEnd
@ Li, aCol[1] PSay cCancel
Return
Endif

@ Li, 000 PSay Replicate("=",Limite)
Li ++
@ Li, aCol[1]  PSay "TOTAL GERAL....."
@ Li, aCol[8]  PSay nT_Valor PICTURE "@E 99,999,999,999.99"
@ Li, aCol[9]  PSay nT_Pago   PICTURE "@E 99,999,999,999.99"
@ Li, aCol[10] PSay nT_Saldo PICTURE "@E 99,999,999,999.99"

If Li <> 80
Roda(cbCont,cbTxt,Tamanho)
Endif

//+-----
-----
```

```

//| Gera arquivo do tipo .DBF com extensão .XLS p/ usuário abrir no
Excel
//+-----
-----
cArqExcel := __RELDIR+NomeProg+"_"+Substr(cUsuario,7,4)+".XLS"
Copy To &cArqExcel

#IFNDEF TOP
dbSelectArea(cAliasImp)
RetIndex(cAliasImp)
Set Filter To
#ELSE
dbSelectArea(cAliasImp)
dbCloseArea()
#endif
dbSetOrder(1)
dbGoTop()

If aReturn[5] == 1
Set Printer TO
dbCommitAll()
OurSpool(wnrel)
Endif

//+-----
//| Abrir planilha MS-Excel
//+-----
If mv_par08 == 1
__CopyFile(cArqExcel,"c:\\"+NomeProg+"_"+Substr(cUsuario,7,4)+".XLS")
If ! ApOleClient("MsExcel")
    MsgBox("MsExcel não instalado")
    Return
Endif
oExcelApp := MsExcel():New()
oExcelApp:WorkBooks:Open(
"c:\\"+NomeProg+"_"+Substr(cUsuario,7,4)+".XLS" )
oExcelApp:SetVisible(.T.)
Endif

Ms_Flush()

Return

```

### **Função para gerar o grupo de parâmetros no SX1**

```

//+-----
+
//| Rotina | CriaSX1 | Autor | Robson Luiz (rleg)| Data | 01.01.07 |

```

```

//+-----
+
//| Descr. | Rotina para criar o grupo de parâmetros.
|
//+-----
+
//| Uso      | Para treinamento e capacitação.
|
//+-----
+
Static Function CriaSx1()
Local aP := {}
Local i := 0
Local cSeq
Local cMvCh
Local cMvPar
Local aHelp := {}

*****
Parâmetros da função padrão
-----
PutSX1(cGrupo,;cOrdem,;
cPergunt,cPerSpa,cPerEng,;
cVar,;
cTipo,;
nTamanho,;
nDecimal,;
nPresel,;
cGSC,;
cValid,;
cF3,;
cGrpSxg,;
cPyme,;
cVar01,;
cDef01,cDefSpa1,cDefEng1,;
cCnt01,;
cDef02,cDefSpa2,cDefEng2,;
cDef03,cDefSpa3,cDefEng3,;
cDef04,cDefSpa4,cDefEng4,;
cDef05,cDefSpa5,cDefEng5,;
aHelpPor,aHelpEng,aHelpSpa,;
cHelp)

Característica do vetor p/ utilização da função SX1
-----
[n,1] --> texto da pergunta
[n,2] --> tipo do dado
[n,3] --> tamanho
[n,4] --> decimal
[n,5] --> objeto G=get ou C=choice

```

```

[n,6] --> validação
[n,7] --> F3
[n,8] --> definição 1
[n,9] --> definição 2
[n,10] -> definição 3
[n,11] -> definição 4
[n,12] -> definição 5
***/
AADD(aP,{"Fornecedor de","C",6,0,"G","","SA2","","","","","",""})
AADD(aP,{"Fornecedor ate","C",6,0,"G","(mv_par02>=mv_par01)","SA2",;,
","","","","","",""})
AADD(aP,{"Tipo de","C",3,0,"G","","05","","","","","",""})
AADD(aP,{"Tipo ate","C",3,0,"G","(mv_par04>=mv_par03)","05",";",
","","","",""})
AADD(aP,{"Vencimento de","D",8,0,"G","","","","","","","",""})
AADD(aP,{"Vencimento ate","D",8,0,"G","(mv_par06>=mv_par05)",";",
","","","","","",""})
AADD(aP,{"Aglutinar pagto.de fornec.","N",1,0,"C","","";,
"Sim","Não","","","",""})
AADD(aP,{"Abrir planilha MS-Excel",;,"N",1,0,"C","","";,
"Sim","Não","","","",""})
AADD(aHelp,{"Informe o código do fornecedor.", "inicial."})
AADD(aHelp,{"Informe o código do fornecedor.", "final."})
AADD(aHelp,{"Tipo de título inicial."})
AADD(aHelp,{"Tipo de título final."})
AADD(aHelp,{"Digite a data do vencimento inicial."})
AADD(aHelp,{"Digite a data do vencimento final."})
AADD(aHelp,{"Aglutinar os títulos do mesmo forne-",;
"cedor totalizando seus valores."})
AADD(aHelp,{"Será gerada uma planilha para ",;
"MS-Excel, abrir esta planilha?"})

For i:=1 To Len(aP)
cSeq := StrZero(i,2,0)
cMvPar := "mv_par"+cSeq
cMvCh := "mv_ch"+IIF(i<=9,Chr(i+48),Chr(i+87))

PutSx1(cPerg,;
cSeq,;
aP[i,1],aP[i,1],aP[i,1],;
cMvCh,;
aP[i,2],;
aP[i,3],;
aP[i,4],;
0,;
aP[i,5],;

```

```

aP[i,6];
aP[i,7];
"";;
"";;
cMvPar;
aP[i,8],aP[i,8],aP[i,8];
"";;
aP[i,9],aP[i,9],aP[i,9];
aP[i,10],aP[i,10],aP[i,10];
aP[i,11],aP[i,11],aP[i,11];
aP[i,12],aP[i,12],aP[i,12];
aHelp[i];
0;
0;
""")
Next i

```

Return



### Exercícios

#### Exercício 14

Implementar um relatório que forneça uma listagem de uma nota fiscal de entrada e seus itens.

## 10. Manipulação de arquivos I

### 2.18 Geração e leitura de arquivos em formato texto

Arquivos do tipo texto (também conhecidos como padrão TXT) são arquivos com registros de tamanho variável. A indicação do final de cada registro é representada por dois bytes, “0D 0A” em hexadecimal ou “13 10” em decimal ou, ainda, “CR LF” para padrão ASCII.

Apesar do tamanho dos registros ser variável, a maioria dos sistemas gera este tipo de arquivo com registros de tamanho fixo, de acordo com um layout específico que indica quais são os dados gravados.

Para ilustrar estes procedimentos, serão gerados arquivos textos, com duas famílias de funções:

**1ª Família:** nesta família serão utilizadas as funções: FCreate(), FWrite(), FCose(), FSeek(), FOpen() e FRead().

**2ª) Família:** nesta família serão utilizadas as funções: FT\_FUse(), FT\_FGoTop(), FT\_FLastRec(), FT\_FEof(), FT\_FReadLn(), FT\_FSkip(), FT\_FGoto(), FT\_FRecno().

---



*Importante*

A diferença entre as duas famílias está na leitura do arquivo texto.

Quando se tratar de arquivo texto com tamanho fixo das linhas, poderão ser utilizadas as duas famílias para leitura do arquivo. Porém, quando se tratar de arquivo texto com tamanho variável das linhas, somente poderá ser utilizada a segunda família, representada pelas funções: FT\_FUse(), FT\_FGoTo(), FT\_FRecno(), FT\_FGoTop(), FT\_FLastRec(), FT\_FEof(), FT\_FReadLn() e FT\_FSkip().

---

### 2.18.1 1ª Família de funções de gravação e leitura de arquivos texto

#### 2.18.1.1 FCREATE()

---

Função de baixo-nível que permite a manipulação direta dos arquivos textos como binários. Ao ser executada FCREATE() cria um arquivo ou elimina o seu conteúdo, e retorna o handle (manipulador) do arquivo, para ser usado nas demais funções de manutenção de arquivo. Após ser utilizado, o Arquivo deve ser fechado através da função FCLOSE().

Na tabela abaixo, estão descritos os atributos para criação do arquivo , definidos no arquivo header fileio.ch

### Atributos definidos no include FileIO.ch

Constante	Valor	Descrição
FC_NORMAL	0	Criação normal do Arquivo (default/padrão).
FC_READONLY	1	Cria o arquivo protegido para gravação.
FC_HIDDEN	2	Cria o arquivo como oculto.
FC_SYSTEM	4	Cria o arquivo como sistema.

Caso desejemos especificar mais de um atributo, basta somá-los. Por exemplo , para criar um arquivo protegido contra gravação e escondido , passamos como atributo  
FC\_READONLY + FC\_HIDDEN..

**Nota:** Caso o arquivo já exista, o conteúdo do mesmo será ELIMINADO, e, seu tamanho será truncado para 0 ( ZERO ) bytes.

**Sintaxe: FCREATE ( < cArquivo > , [ nAtributo ] )**

**Parâmetros:**

<b>cArquivo</b>	Nome do arquivo a ser criado, podendo ser especificado um pacote (patch) absoluto ou relativo, para criar arquivos no ambiente local ( Remote ) ou no Servidor, respectivamente .
<b>nAtributo</b>	Atributos do arquivo a ser criado (Vide Tabela de atributos abaixo). Caso não especificado, o DEFAULT é FC_NORMAL.

**Retorno:**

<b>Numérico</b>	A função retornará o Handle do arquivo para ser usado nas demais funções de manutenção de arquivo. O Handle será maior ou igual a zero. Caso não seja possível criar o arquivo, a função retornará o handle -1 , e será possível obter maiores detalhes da ocorrência através da função FERROR() .
-----------------	--

#### 2.18.1.2 FWRITE()

Função que permite a escrita em todo ou em parte do conteúdo do *buffer*, limitando a quantidade de Bytes através do parâmetro nQtdBytes. A escrita começa a partir da posição corrente do ponteiro de arquivos, e a função FWRITE retornará a quantidade real de bytes escritos. Através das funções FOPEN(), FCREATE(), ou FOPENPORT(), podemos abrir ou criar um arquivo ou abrir uma porta de comunicação, para o qual serão gravados ou enviados os dados do *buffer* informado. Por tratar-se de uma função de manipulação de conteúdo binário, são suportados na String cBuffer todos os caracteres da tabela ASCII, inclusive caracteres de controle (ASC 0 , ASC 12 , ASC 128 , etc.).

Caso aconteça alguma falha na gravação, a função retornará um número menor que o nQtdBytes. Neste caso, a função FERROR() pode ser utilizada para determinar o erro específico ocorrido. A gravação no arquivo é realizada a partir da posição atual do ponteiro, que pode ser ajustado através das funções FSEEK() , FREAD() ou FREADSTR().

**Sintaxe:** FWRITE ( < nHandle > , < cBuffer > , [ nQtdBytes ] )

**Parâmetros:**

<b>nHandle</b>	É o manipulador de arquivo ou <i>device</i> retornado pelas funções FOPEN(), FCREATE(), ou FOPENPORT().
<b>cBuffer</b>	<cBuffer> é a cadeia de caracteres a ser escrita no arquivo especificado. O tamanho desta variável deve ser maior ou igual ao tamanho informado em nQtdBytes (caso seja informado o tamanho).
<b>nQtdBytes</b>	<nQtdBytes> indica a quantidade de bytes a serem escritos a partir da posição corrente do ponteiro de arquivos. Caso seja omitido, todo o conteúdo de <cBuffer> é escrito.

**Retorno:**

<b>Numérico</b>	FWRITE() retorna a quantidade de bytes escritos na forma de um valor numérico inteiro. Caso o valor retornado seja igual a <nQtdBytes>, a operação foi bem sucedida. Caso o valor de retorno seja menor que <nBytes> ou zero, ou o disco está cheio ou ocorreu outro erro. Neste caso, utilize a função FERROR() para obter maiores detalhes da ocorrência.
-----------------	---

### 2.18.1.3 FCLOSE()

---

Função de tratamento de arquivos de baixo nível utilizada para fechar arquivos binários e forçar que os respectivos *buffers* do DOS sejam escritos no disco. Caso a operação falhe, FCLOSE() retorna falso (.F.). FERROR() pode, então, ser usado para determinar a razão exata da falha. Por exemplo, ao tentar usar FCLOSE() com um *handle* (tratamento dado ao arquivo pelo sistema operacional) inválido retorna falso (.F.) e FERROR() retorna erro 6 do DOS,       *invalid handle*. Consulte FERROR() para obter uma lista completa dos códigos de erro.

**Nota:** Esta função permite acesso de baixo nível aos arquivos e dispositivos do DOS. Ela deve ser utilizada com extremo cuidado e exige que se conheça a fundo o sistema operacional utilizado.

**Sintaxe:** FCLOSE ( < nHandle > )

**Parâmetros:**

<b>nHandle</b>	<i>Handle</i> do arquivo obtido previamente através de FOPEN() ou FCREATE().
----------------	--

**Retorno:**

<b>Lógico</b>	Retorna falso (.F.) se ocorre um erro enquanto os <i>buffers</i> estão sendo escritos; do contrário, retorna verdadeiro (.T.).
---------------	--

#### 2.18.1.4 FSEEK()

Função que posiciona o ponteiro do arquivo para as próximas operações de leitura ou gravação. As movimentações de ponteiros são relativas à nOrigem que pode ter os seguintes valores, definidos em fileio.ch:

**Tabela A: Origem a ser considerada para a movimentação do ponteiro de posicionamento do Arquivo.**

Origem	Constate(fileio.ch)	Operação
0	FS_SET	Ajusta a partir do inicio do arquivo. (Default).
1	FS_RELATIVE	Ajuste relativo a posição atual do arquivo.
2	FS_END	Ajuste a partir do final do arquivo.

**Sintaxe: FSEEK ( < nHandle > , [ nOffSet ] , [ nOrigem ] )**

**Parâmetros:**

<b>nHandle</b>	Manipulador obtido através das funções FCREATE, FOPEN.
<b>nOffSet</b>	nOffSet corresponde ao número de <i>bytes</i> no ponteiro de posicionamento do arquivo a ser movido. Pode ser um número positivo, zero ou negativo, a ser considerado a partir do parâmetro passado em nOrigem.
<b>nOrigem</b>	Indica a partir de qual posição do arquivo, o nOffset será considerado.

**Retorno:**

<b>Numérico</b>	FSEEK() retorna a nova posição do ponteiro de arquivo com relação ao início do arquivo (posição 0) na forma de um valor numérico inteiro. Este valor não leva em conta a posição original do ponteiro de arquivos antes da execução da função FSEEK().
-----------------	--

#### 2.18.1.5 FOPEN()

Função de tratamento de arquivo de baixo nível que abre um arquivo binário existente para que este possa ser lido e escrito, dependendo do argumento <nModo>. Toda vez que houver um erro na abertura do arquivo, FERROR() pode ser usado para retornar o código de erro do Sistema Operacional. Por exemplo, caso o arquivo não exista, FOPEN() retorna -1 e FERROR() retorna 2 para indicar que o arquivo não foi encontrado. Veja FERROR() para uma lista completa dos códigos de erro.

Caso o arquivo especificado seja aberto, o valor retornado é o handle (manipulador) do Sistema Operacional para o arquivo. Este valor é semelhante a um alias no sistema de banco de dados, e ele é exigido para identificar o arquivo aberto para as outras funções de tratamento de arquivo. Portanto, é importante sempre atribuir o valor que foi retornado a uma variável para uso posterior, como mostra o exemplo desta função.

**Sintaxe: FOPEN ( < cArq > , [ nModo ] )**

**Parâmetros:**

cArq	Nome do arquivo a ser aberto que inclui o pacote, caso haja um.
nModo	Modo de acesso DOS solicitado que indica como o arquivo aberto deve ser acessado. O acesso é de uma das categorias relacionadas na Tabela A e as restrições de compartilhamento relacionada na Tabela B. O modo padrão é zero, somente para leitura, com compartilhamento por Compatibilidade. Ao definirmos o modo de acesso, devemos somar um elemento da Tabela A com um elemento da Tabela B.

**Retorno:**

Numérico	FOPEN() retorna o <i>handle</i> de arquivo aberto na faixa de zero a 65.535. Caso ocorra um erro, FOPEN() retorna -1.
----------	---

#### **2.18.1.6      FREAD()**

---

Função que realiza a leitura dos dados a partir um arquivo aberto, através de FOPEN(), FCREATE() e/ou FOPENPORT(), e armazena os dados lidos por referência no *buffer* informado.

FREAD() lerá até o número de *bytes* informado em nQtdBytes; caso aconteça algum erro ou o arquivo chegue ao final, FREAD() retornará um número menor que o especificado em nQtdBytes. FREAD() lê normalmente caracteres de controle (ASC 128, ASC 0, etc.) e lê a partir da posição atual do ponteiro atual do arquivo, que pode ser ajustado ou modificado pelas funções FSEEK() , FWRITE() ou FREADSTR().

A variável String a ser utilizada como *buffer* de leitura deve ser sempre pré-alocada e passada como referência. Caso contrário, os dados não poderão ser retornados.

**Sintaxe: FREAD ( < nHandle > , < cBuffer > , < nQtdBytes > )**

**Parâmetros:**

<b>nHandle</b>	É o manipulador (Handle) retornado pelas funções FOPEN(), FCREATE(), FOPENPORT(), que faz referência ao arquivo a ser lido.
<b>cBuffer</b>	É o nome de uma variável do tipo String , a ser utilizada como <i>buffer</i> de leitura, onde os dados lidos deverão ser armazenados. O tamanho desta variável deve ser maior ou igual ao tamanho informado em nQtdBytes. Esta variável deve ser sempre passada como referência. (@ antes do nome da variável), caso contrário os dados lidos não serão retornados.
<b>nQtdBytes</b>	Define a quantidade de <i>bytes</i> que devem ser lidas do arquivo a partir posicionamento do ponteiro atual.

**Retorno:**

<b>Numérico</b>	Quantidades de <i>bytes</i> lidos. Caso a quantidade seja menor que a solicitada, isto indica erro de leitura ou final de arquivo. Verifique a função FERROR() para mais detalhes.
-----------------	--

**Exemplo: Geração de arquivo TXT, utilizando a primeira família de funções**

```
#include "protheus.ch"

/*
+-----
| Programa      | GeraTXT   | Autor | SERGIO FUZINAKA | Data | |
+-----
| Descrição      | Gera o arquivo TXT, a partir do Cadastro de
Clientes
+-----
| Uso           | Curso ADVPL
+-----
*/
User Function GeraTXT()

//+-----+
//| Declaração de Variáveis      |
//+-----+
Local oGeraTxt
Private cPerg      := "EXPSA1"
Private cAlias     := "SA1"
//CriaSx1(cPerg)
```

```

//Pergunte(cPerg,.F.)
dbSelectArea(cAlias)
dbSetOrder(1)

//+-----+
//| Montagem da tela de processamento.| 
//+-----+

DEFINE MSDIALOG oGeraTxt TITLE OemToAnsi("Geração de Arquivo Texto")
;
FROM 000,000 TO 200,400 PIXEL

@ 005,005 TO 095,195 OF oGeraTxt PIXEL
@ 010,020 Say " Este programa ira gerar um arquivo texto, conforme
os parame- ";
OF oGeraTxt PIXEL
@ 018,020 Say " tros definidos pelo usuário, com os registros do
arquivo de ";
OF oGeraTxt PIXEL
@ 026,020 Say " SA1 " OF oGeraTxt PIXEL

DEFINE SBUTTON FROM 070, 030 TYPE 1 ;
ACTION (OkGeraTxt(),oGeraTxt:End()) ENABLE OF oGeraTxt

DEFINE SBUTTON FROM 070, 070 TYPE 2 ;
ACTION (oGeraTxt:End()) ENABLE OF oGeraTxt

DEFINE SBUTTON FROM 070, 110 TYPE 5 ;
ACTION (Pergunte(cPerg,.T.)) ENABLE OF oGeraTxt

ACTIVATE DIALOG oGeraTxt CENTERED

Return Nil

```

**Exemplo (continuação):**

```

*/
+-----+
----+
| Função      | OKGERATXT | Autor | SERGIO FUZINAKA | Data |
|             |
+-----+
----+
| Descrição      | Função chamada pelo botão OK na tela inicial de
processamento. |
|                 | Executa a geração do arquivo texto.
|                 |
+-----+
----+
/*

```

```

Static Function OkGeraTxt

//+-----
+-----+
//| Cria o arquivo texto
//+-----+
+-----+
Private cArqTxt := "\SYSTEM\EXPSA1.TXT"
Private nHdl      := fCreate(cArqTxt)

If nHdl == -1
    MsgBox("O arquivo de nome "+cArqTxt+" não pode ser executado!
Verifique os parâmetros.", "Atenção!")
    Return
Endif

// Inicializa a régua de processamento
Processa({| | RunCont() },"Processando...")

Return Nil

/*
+-----+
+-----+
| Função      | RUNCONT  | Autor | SERGIO FUZINAKA | Data |
|             |          |       |               |       |
+-----+
+-----+
| Descrição      | Função auxiliar chamada pela PROCESSA. A função
PROCESSA          |
|                 | monta a janela com a régua de processamento.
|             |       |
+-----+
+-----+
*/
Static Function RunCont

Local cLin

dbSelectArea(cAlias)
dbGoTop()
ProcRegua(RecCount()) // Numero de registros a processar

While (cAlias)->(!EOF())
//Incrementa a régua
IncProc()

```

```

cLin := (cAlias)->A1_FILIAL
cLin += (cAlias)->A1_COD
cLin += (cAlias)->A1_LOJA
    cLin += (cAlias)->A1_NREDUZ
    cLin += STRZERO((cAlias)->A1_MCOMPRA*100,16) // 14,2
    cLin += DTOS((cAlias)->A1_ULTCOM)//AAAAAMMDD
    cLin += CRLF

```

**Exemplo (continuação):**

```

//+-----
--+
//| Gravação no arquivo texto. Testa por erros durante a gravação da
|
//| linha montada.
|
//+-----
--+
If fWrite(nHdl,cLin,Len(cLin)) != Len(cLin)
    If !MsgAlert("Ocorreu um erro na gravação do
arquivo."+
                    "Continua?","Atenção!")
        Exit
    Endif
Endif

(cAlias)->(dbSkip())
EndDo

// O arquivo texto deve ser fechado, bem como o dialogo criado na
função anterior
fClose(nHdl)

Return Nil

```

Note que para a geração do arquivo TXT foram utilizadas, basicamente, as funções FCreate, FWrite e FClose que, respectivamente, gera o arquivo, adiciona dados e fecha o arquivo. No exemplo, o formato é estabelecido pela concatenação dos dados na variável **cLin** a qual é utilizada na gravação dos dados. Para a leitura de dados TXT serão utilizada as funções FOpen e FRead.

**Exemplo: Leitura de arquivo TXT, utilizando a primeira família de funções**

```
#Include "protheus.ch"
```

```

/*
+-----
| Programa | LeTXT           | Autor | SERGIO FUZINAKA | Data |
|          |                         |
+-----+
| Descrição      | Leitura de arquivo TXT
|          |                         |
+-----+
| Uso           | Curso ADVPL
|          |                         |
+-----+
|          |                         |
+-----+
/*
User Function LeTXT()

//+-----
--+ Declaração de Variáveis
|
//+-----
--+
Local cPerg           := "IMPSA1"
Local oLeTxt

Private cAlias := "SA1"

//CriaSx1(cPerg)
//Pergunte(cPerg,F.)

```

**Exemplo (continuação):**

```

dbSelectArea(cAlias)
dbSetOrder(1)

//+-----
--+ Montagem da tela de processamento
|
//+-----
--+
DEFINE MSDIALOG oLeTxt TITLE OemToAnsi("Leitura de Arquivo Texto");
FROM 000,000 TO 200,400 PIXEL
@ 005,005 TO 095,195 OF oLeTxt PIXEL

```

```

@ 10,020 Say " Este programa ira ler o conteúdo de um arquivo texto,
conforme";
OF oLeTxt PIXEL
@ 18,020 Say " os parâmetros definidos pelo usuário, com os
registros do arquivo";
OF oLeTxt PIXEL
@ 26,020 Say " SA1" OF oLeTxt PIXEL

DEFINE SBUTTON FROM 070, 030 TYPE 1 ;
ACTION (OkLeTxt(),oLeTxt:End()) ENABLE OF oLeTxt

DEFINE SBUTTON FROM 070, 070 TYPE 2 ;
ACTION (oLeTxt:End()) ENABLE OF oLeTxt

DEFINE SBUTTON FROM 070, 110 TYPE 5 ;
ACTION (Pergunte(cPerg,.T.)) ENABLE OF oLeTxt
ACTIVATE DIALOG oLeTxt CENTERED

Return Nil

*/
+-----
----+
| Função      | OKLEXTXT    | Autor | SERGIO FUZINAKA | Data |
|             |              |       |
+-----+
----+
| Descrição      | Função chamada pelo botão OK na tela inicial de
processamento | |
|             | Executa a leitura do arquivo texto
|             |
+-----+
----+
/*
Static Function OkLeTxt()

//+
---+
//| Abertura do arquivo texto
|
//+
---+
Private cArqTxt := "\SYSTEM\EXPSA1.TXT"
Private nHdl     := fOpen(cArqTxt,68)

If nHdl == -1
    MsgBox("O arquivo de nome "+cArqTxt+" não pode ser aberto!
Verifique os parâmetros.", "Atenção!")
    Return

```

```

Endif

// Inicializa a régua de processamento
Processa({| | RunCont() },"Processando...")
Return Nil

/*
+-----
----+
| Função      | RUNCONT     | Autor | SERGIO FUZINAKA | Data |
|             |             |       |
+-----+
----+
| Descrição      | Função auxiliar chamada pela PROCESSA. A função
PROCESSA          |
|                 | monta a janela com a régua de processamento.
|             |
+-----+
----+
*/
Static Function RunCont

Local nTamFile    := 0
Local nTamLin     := 56
Local cBuffer      := ""
Local nBtLidos    := 0
Local cFilSA1      := ""
Local cCodSA1      := ""
Local cLojaSA1     := ""

//123456789012345678901234567890123456789012345678901234567890123456
7890
//000000000100000000200000000300000000400000000500000000600000
00070
//FFCCCCCLNNNNNNNNNNNNNNNNNNNNNNNNNNNNVVVVVVVVVVVVVVVVVVVDDDDDDDD
//A1_FILIAL           - 01, 02 - TAM: 02
//A1_COD               - 03, 08 - TAM: 06
//A1_LOJA              - 09, 10 - TAM: 02
//A1_NREDUZ             - 11, 30 - TAM: 20
//A1_MCOMPRA            - 31, 46 - TAM: 14,2
//A1_ULTCOM             - 47, 54 - TAM: 08

nTamFile := fSeek(nHdI,0,2)
fSeek(nHdI,0,0)
cBuffer := Space(nTamLin) // Variável para criação da linha do
registro para leitura

ProcRegua(nTamFile) // Numero de registros a processar
While nBtLidos < nTamFile

```

```

//Incrementa a régua
IncProc()

// Leitura da primeira linha do arquivo texto
nBtLidos += fRead(nHdl,@cBuffer,nTamLin)

cFilSA1      := Substr(cBuffer,01,02) // 01, 02 - TAM: 02
cCodSA1      := Substr(cBuffer,03,06) // 03, 08 - TAM: 06
cLojaSA1     := Substr(cBuffer,09,02) // 09, 10 - TAM: 02

While .T.
    IF dbSeek(cFilSA1+cCodSA1+cLojaSA1)
        cCodSA1 := SOMA1(cCodSA1)
        Loop
    Else
        Exit
    Endif
Enddo

```

**Exemplo (continuação):**

```

dbSelectArea(cAlias)
RecLock(cAlias,.T.)
(cAlias)->A1_FILIAL      := cFilSA1 // 01, 02 - TAM: 02
(cAlias)->A1_COD         := cCodSA1 // 03, 08 - TAM: 06
(cAlias)->A1_LOJA        := cLojaSA1 // 09, 10 - TAM: 02
(cAlias)->A1_NREDUZ      := Substr(cBuffer,11,20)
// 11, 30 - TAM: 20
(cAlias)->A1_MCOMPRA     := Val(Substr(cBuffer,31,16))/100
// 31, 46 - TAM: 14,2
(cAlias)->A1_ULTCOM      := STOD(Substr(cBuffer,47,08))
// 47, 54 - TAM: 08
MSUnLock()

EndDo

// O arquivo texto deve ser fechado, bem como o dialogo criado na
// função anterior.
fClose(nHdl)

Return Nil

```

## 2.18.2 2ª Família de funções de gravação e leitura de arquivos texto

### 2.18.2.1 FT\_FUSE()

Função que abre ou fecha um arquivo texto para uso das funções FT\_F\*. As funções FT\_F\* são usadas para ler arquivos texto, onde as linhas são delimitadas pela seqüência de caracteres CRLF ou LF (\*) e o tamanho máximo de cada linha é 1022 bytes. O arquivo é aberto em uma área de trabalho, similar à usada pelas tabelas de dados.

**Sintaxe:** `FT_FUSE ( [ cTXTFile ] )`

**Parâmetros:**

<b>cTXTFile</b>	Corresponde ao nome do arquivo TXT a ser aberto. Caso o nome não seja passado, e já exista um arquivo aberto, o mesmo é fechado.
-----------------	--

**Retorno:**

<b>Numérico</b>	A função retorna o <i>Handle</i> de controle do arquivo. Em caso de falha de abertura, a função retornará -1
-----------------	--

#### **2.18.2.2      `FT_FGOTOP()`**

---

A função tem como objetivo mover o ponteiro, que indica a leitura do arquivo texto, para a posição absoluta especificada pelo argumento <nPos>.

**Sintaxe:** `FT_FGOTO ( <nPos> )`

**Parâmetros:**

<b>nPos</b>	Indica a posição que será colocado o ponteiro para leitura dos dados no arquivo.
-------------	--

#### **2.18.2.3      `FT_FLASTREC()`**

---

Função que retorna o número total de linhas do arquivo texto aberto pela FT\_FUse. As linhas são delimitadas pela seqüência de caracteres CRLF ou LF.

**Sintaxe:** `FT_FLASTREC( )`

**Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

**Retorno:**

<b>Numérico</b>	Retorna a quantidade de linhas existentes no arquivo. Caso o arquivo esteja vazio, ou não exista arquivo aberto, a função retornará 0 (zero).
-----------------	---

## 2.18.2.4 FT\_FEOF()

---

Função que retorna verdadeiro (.t.) se o arquivo texto aberto pela função FT\_FUSE() estiver posicionado no final do arquivo, similar à função EOF() utilizada para arquivos de dados.

**Sintaxe:** FT\_FEOF( )

**Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

**Retorno:**

<b>Lógico</b>	Retorna <i>true</i> caso o ponteiro do arquivo tenha chegado ao final; <i>false</i> caso contrário.
---------------	---

## 2.18.2.5 FT\_FREADLN()

---

Função que retorna uma linha de texto do arquivo aberto pela FT\_FUse. As linhas são delimitadas pela seqüência de caracteres CRLF ( *chr(13) + chr(10)* ), ou apenas LF ( *chr(10)* ), e o tamanho máximo de cada linha é 1022 bytes.

**Sintaxe:** FT\_FREADLN( )

**Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

**Retorno:**

<b>Caracter</b>	Retorna a linha inteira na qual está posicionado o ponteiro para leitura de dados.
-----------------	--

## 2.18.2.6 FT\_FSKIP()

---

Função que move o ponteiro do arquivo texto aberto pela FT\_FUSE() para a próxima linha, similar ao DBSKIP() usado para arquivos de dados.

**Sintaxe:** FT\_FSKIP ( [ nLinhas ] )

**Parâmetros:**

<b>nLinhas</b>	nLinhas corresponde ao número de linhas do arquivo TXT ref. movimentação do ponteiro de leitura do arquivo.
----------------	---

**Retorno**

<b>Nenhum</b>	()
---------------	----

## 2.18.2.7 FT\_FGOTO()

---

Função utilizada para mover o ponteiro, que indica a leitura do arquivo texto, para a posição absoluta especificada pelo argumento <nPos>.

**Sintaxe:** FT\_FGOTO ( <nPos> )

**Parâmetros:**

<b>nPos</b>	Indica a posição que será colocado o ponteiro para leitura dos dados no arquivo.
-------------	--

**Retorno:**

<b>Nenhum</b>	()
---------------	----

## 2.18.2.8 FT\_FRECNO()

---

A função tem o objetivo de retornar a posição do ponteiro do arquivo texto.

A função FT\_FRecno retorna a posição corrente do ponteiro do arquivo texto aberto pela FT\_FUse.

**Sintaxe:** FT\_FRECNO ( )

**Parâmetros:**

<b>Nenhum</b>	()
---------------	----

**Retorno:**

<b>Caracter</b>	Retorna a posição corrente do ponteiro do arquivo texto.
-----------------	--

### Exemplo: Leitura de arquivo TXT, utilizando a segunda família de funções

```
#Include "Protheus.ch"

/*
+-----
| Programa | LeArqTXT | Autor | Robson Luiz | Data |
+-----
| Descrição | Leitura de arquivo TXT |
+-----
| Uso       | Curso ADVPL |
+-----
----*/
User Function LeArqTxt()

Private nOpc          := 0
Private cCadastro     := "Ler arquivo texto"
Private aSay           := {}
Private aButton         := {}

AADD( aSay, "O objetivo desta rotina é efetuar a leitura em um
arquivo texto" )

AADD( aButton, { 1,.T.,{} | nOpc := 1,FechaBatch()} )
AADD( aButton, { 2,.T.,{} | FechaBatch() } )

FormBatch( cCadastro, aSay, aButton )

If nOpc == 1
    Processa( {} | Import(), "Processando..." )
Endif
Return Nil

//+
//| Função - Import()
//+
Static Function Import()

Local cBuffer      := ""
Local cFileOpen    := ""
Local cTitulo1     := "Selecione o arquivo"
Local cExtens      := "Arquivo TXT | *.txt"

/***
```

```

* -----
* cGetFile(<ExpC1>,<ExpC2>,<ExpN1>,<ExpC3>,<ExpL1>,<ExpN2>)
* -----
* <ExpC1> - Expressão de filtro
* <ExpC2> - Titulo da janela
* <ExpN1> - Numero de mascara default 1 para *.Exe
* <ExpC3> - Diretório inicial se necessário
* <ExpL1> - .F. botão salvar - .T. botão abrir
* <ExpN2> - Mascara de bits para escolher as opções de visualização
do objeto
* (prconst.ch)
*/
cFileOpen := cGetFile(cExtens,cTitulo1,,cMainPath,.T.)

If !File(cFileOpen)
    MsgBox("Arquivo texto: "+cFileOpen+" não localizado",cCadastro)
    Return
Endif

FT_FUSE(cFileOpen) //ABRIR
FT_FGOTOP() //PONTO NO TOPO
ProcRegua(FT_FLASTREC()) //QTOS REGISTROS LER

While !FT_FEOF() //FACA ENQUANTO NAO FOR FIM DE ARQUIVO
    IncProc()

    // Capturar dados
    cBuffer := FT_FREADLN() //LEND0 LINHA

    cMsg := "Filial: " +SubStr(cBuffer,01,02) +
Chr(13)+Chr(10)
    cMsg += "Código: " +SubStr(cBuffer,03,06) +
Chr(13)+Chr(10)
    cMsg += "Loja: " +SubStr(cBuffer,09,02) +
Chr(13)+Chr(10)
    cMsg += "Nome fantasia: " +SubStr(cBuffer,11,15) +
Chr(13)+Chr(10)
    cMsg += "Valor: " +SubStr(cBuffer,26,14) +
Chr(13)+Chr(10)
    cMsg += "Data: " +SubStr(cBuffer,40,08) +
Chr(13)+Chr(10)

    MsgBox(cMsg)

    FT_FSKIP() //próximo registro no arquivo txt
EndDo

Exemplo (continuação):

FT_FUSE() //fecha o arquivo txt
MsgInfo("Processo finalizada")

```

Return Nil



### Exercícios

#### Exercício 15

Desenvolver uma rotina que realize a exportação dos itens marcados no cadastro de clientes para um arquivo TXT em um diretório especificado pelo usuário.

#### Exercício 16

Desenvolver uma rotina que realize a importação dos dados de clientes contidos em um arquivo TXT, sendo o mesmo selecionado pelo usuário.

## 11. Oficina de programação I

### 2.19 Interfaces com sintaxe clássica

A sintaxe convencional para definição de componentes visuais da linguagem ADVPL depende diretamente do *include* especificado no cabeçalho do fonte. Os dois *includes* disponíveis para o ambiente ADVPL Protheus são:

- RWMAKE.CH: permite a utilização da sintaxe CLIPPER na definição dos componentes visuais.
- PROTHEUS.CH: permite a utilização da sintaxe ADVPL convencional, a qual é um aprimoramento da sintaxe CLIPPER, com a inclusão de novos atributos para os componentes visuais disponibilizados no ERP Protheus.

Para ilustrar a diferença na utilização destes dois *includes*, seguem abaixo as diferentes definições para o componentes Dialog e MsDialog:

#### Exemplo 01 – Include Rwmake.ch

```
#include "rwmakch.ch"  
  
@ 0,0 TO 400,600 DIALOG oDlg TITLE "Janela em sintaxe Clipper"  
ACTIVATE DIALOG oDlg CENTERED
```

#### Exemplo 02 – Include Protheus.ch

```
#include "protheus.ch"

DEFINE MSDIALOG oDlg TITLE "Janela em sintaxe ADVPL "FROM 000,000 TO
400,600 PIXEL
ACTIVATE MSDIALOG oDlg CENTERED
```



*Importante*

Ambas as sintaxes produzirão o mesmo efeito quando compiladas e executadas no ambiente Protheus. Mas, deve ser utilizada a sintaxe ADVPL através do uso do include PROTHEUS.CH

Os componentes da interface visual que serão tratados neste tópico, utilizando a sintaxe clássica da linguagem ADVPL são:

- ❑ **BUTTON()**
- ❑ **CHECKBOX()**
- ❑ **COMBOBOX()**
- ❑ **FOLDER()**
- ❑ **MSDIALOG()**
- ❑ **MSGET()**
- ❑ **RADIO()**
- ❑ **SAY()**
- ❑ **SBUTTON()**

Executar o fonte DIALOG\_OBJETOS.PRW e avaliar a definição dos componentes utilizados utilizando a sintaxe clássica.

### BUTTON()

<b>Sintaxe</b>	<b>@ nLinha,nColuna BUTTON cTexto SIZE nLargura,nAltura UNIDADE OF oObjetoRef ACTION AÇÃO</b>
<b>Descrição</b>	Define o componente visual Button, que permite a inclusão de botões de operação na tela da interface, os quais serão visualizados somente com um texto simples para sua identificação.

### CHECKBOX()

<b>Sintaxe</b>	<b>@ nLinha,nColuna CHECKBOX oCheckBox VAR VARIABEL PROMPT cTexto</b>
----------------	---

	<b>WHEN WHEN UNIDADE OF oObjetoRef SIZE nLargura,nAltura MESSAGE cMensagem</b>	
<b>Descrição</b>	Define o componente visual CheckBox, o qual permite a utilização da uma marca para habilitar ou não uma opção escolhida, sendo esta marca acompanhada de um texto explicativo. Difere do RadioMenu pois cada elemento do check é único, mas o Radio permite a utilização de uma lista junto com um controle de seleção.	

#### COMBOBOX()

---

<b>Sintaxe</b>	<b>@ nLinha,nColuna COMBOBOX VARIABEL ITEMS AITENS SIZE nLargura,nAltura UNIDADE OF oObjetoRef</b>
<b>Descrição</b>	Define o componente visual ComboBox, o qual permite seleção de um item dentro de uma lista de opções de textos simples no formato de um vetor.

#### FOLDER()

---

<b>Sintaxe</b>	<b>@ nLinha,nColuna FOLDER oFolder OF oObjetoRef PROMPT &amp;cTexto1,...,&amp;cTextoX PIXEL SIZE nLargura,nAltura</b>
<b>Descrição</b>	Define o componente <i>Visual Folder</i> , o qual permite a inclusão de diversos <i>Dialogs</i> dentro de uma mesma interface visual. Um <i>folder</i> pode ser entendido como um <i>array de dialogs</i> , aonde cada painel recebe seus componentes e tem seus atributos definidos independentemente dos demais.

#### MSDIALOG()

---

<b>Sintaxe</b>	<b>DEFINE MSDIALOG oObjetoDLG TITLE cTitulo FROM nLinIni,nColIni TO nLiFim,nColFim OF oObjetoRef UNIDADE</b>
<b>Descrição</b>	Define o componente MSDIALOG(), o qual é utilizado como base para os demais componentes da interface visual, pois um componente MSDIALOG() é uma janela da aplicação.

#### MSGET()

---

<b>Sintaxe</b>	<b>@ nLinha, nColuna MSGET VARIABEL SIZE nLargura,nAltura UNIDADE OF oObjetoRef F3 cF3 VALID VALID WHEN WHEN PICTURE cPicture</b>
<b>Descrição</b>	Define o componente visual MSGET, o qual é utilizado para captura de informações digitáveis na tela da interface.

#### RADIO()

---

<b>Sintaxe</b>	<b>@ nLinha,nColuna RADIO oRadio VAR nRadio 3D SIZE nLargura,nAltura &lt;ITEMS PROMPT&gt; cItem1,cItem2,...,cItemX OF oObjetoRef UNIDADE ON CHANGE CHANGE ON CLICK CLICK</b>
<b>Descrição</b>	Define o componente visual Radio, também conhecido como RadioMenu, o qual é seleção de uma opção ou de múltiplas opções através de uma marca para os

	<p>itens exibidos de uma lista. Difere do componente <i>checkbox</i>, pois cada elemento de <i>check</i> é sempre único, e o Radio pode conter um ou mais elementos.</p>
--	--

## SAY()

---

<b>Sintaxe</b>	<b>@ nLinha, nColuna SAY cTexto SIZE nLargura,nAltura UNIDADE OF oObjetoRef</b>
<b>Descrição</b>	Define o componente visual SAY, o qual é utilizado para exibição de textos em uma tela de interface.

## SBUTTON()

---

<b>Sintaxe</b>	<b>DEFINE SBUTTON FROM nLinha, nColuna TYPE N ACTION AÇÃO STATUS OF oObjetoRef</b>
<b>Descrição</b>	Define o componente visual SButton, que permite a inclusão de botões de operação na tela da interface, os quais serão visualizados, dependendo da interface do sistema ERP utilizada, somente com um texto simples para sua identificação, ou com uma imagem (BitMap) pré-definido.

## 2.20 Régulas de processamento

---

Os indicadores de progresso ou réguas de processamento disponíveis na linguagem ADVPL que serão abordados neste material são:

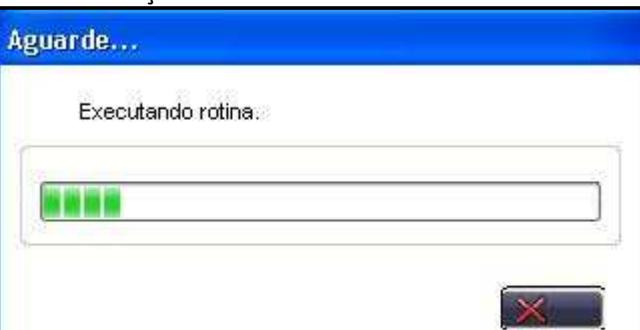
- [RPTSTATUS\(\)](#)
- [PROCESSA\(\)](#)
- [MSNEWPROCESS\(\)](#)
- [MSAGUARDE\(\)](#)
- [MSGRUN\(\)](#)

### 2.20.1 RptStatus()

Régua de processamento simples, com apenas um indicador de progresso, utilizada no processamento de relatórios do padrão SetPrint().

- **Sintaxe: RptStatus(bAcao, cMensagem)**
- **Retorno: Nil**

### Parâmetros:

bAcao	Bloco de código que especifica a ação que será executada com o acompanhamento da régua de processamento.
cMensagem	Mensagem que será exibida na régua de processamento durante a execução.
Aparência	

### Exemplo: Função RPTStatus() e acessórios

```
/*
+-----
| Função      | GRPTSTATUS | Autor | ROBSON LUIZ           | Data |
|             |            |        |                         |
+-----+
| Descrição      | Programa que demonstra a utilização das funções
RPTSTATUS()   |
|             | SETREGUA() E INCREGUA()
|             |
+-----+
| Uso          | Curso ADVPL
|             |
+-----+
*/
User Function GRptStatus()
Local aSay      := {}
Local aButton := {}
Local nOpc      := 0
Local cTitulo := "Exemplo de Funções"
Local cDesc1   := "Este programa exemplifica a utilização da função
Processa() em conjunto"
Local cDesc2   := "com as funções de incremento ProcRegua() e
IncProc()"
```

```

Private cPerg := "RPTSTA"

CriaSX1()
Pergunte(cPerg,.F.)

AADD( aSay, cDesc1 )
AADD( aSay, cDesc2 )

AADD( aButton, { 5, .T., {|| Pergunte(cPerg,.T. ) } } )
AADD( aButton, { 1, .T., {|| nOpc := 1, FechaBatch() } } )
AADD( aButton, { 2, .T., {|| FechaBatch() } } )

FormBatch( cTitulo, aSay, aButton )

If nOpc <> 1
    Return Nil
Endif

RptStatus( {|| IEnd| RunProc(@IEnd)}, "Aguarde...","Executando
rotina.", .T. )

Return Nil

```

#### **Exemplo: Funções acessórias da RPTStatus()**

---

```

/*
+-----
-----+
| Função      | RUNPROC      | Autor | ROBSON LUIZ           | Data |
|             |              |       |                         |
+-----+
-----+
| Descrição      | Função de processamento executada através da
RPTSTATUS()      |
+-----+
-----+
| Uso          | Curso ADVPL
|             |           |
+-----+
-----+
*/

```

Static Function RunProc(IEnd)

Local nCnt := 0

dbSelectArea("SX5")  
dbSetOrder(1)

```

dbSeek(xFilial("SX5")+mv_par01,.T.)

While !Eof() .And. X5_FILIAL == xFilial("SX5") .And. X5_TABELA <=
mv_par02
    nCnt++
    dbSkip()
End

dbSeek(xFilial("SX5")+mv_par01,.T.)

SetRegua(nCnt)
While !Eof() .And. X5_FILIAL == xFilial("SX5") .And. X5_TABELA <=
mv_par02
    IncRegua()
    If !End
        MsgInfo(cCancel,"Fim")
        Exit
    Endif
    dbSkip()
End
Return .T.

```

## **SETREGUA()**

---

A função SetRegua() é utilizada para definir o valor máximo da régua de progressão criada através da função RptStatus().

**Sintaxe:** **SetRegua(nMaxProc)**

**Parâmetros:**

<b>nMaxProc</b>	Variável que indica o valor máximo de processamento (passos) que serão indicados pela régua.
-----------------	--

**Retorno:**

<b>Nenhum</b>	()
---------------	----

**Exemplo:**

```

...
dbSelectArea("SA1")
dbGoTop()
SetRegua>LastRec()
While !Eof()
    IncRegua()

```

If Li > 60

...

## **INCREGUA()**

A função IncRegua() é utilizada para incrementar valor na régua de progressão criada através da função RptStatus()

**Sintaxe:** **IncRegua(cMensagem)**

**Parâmetros:**

<b>cMensagem</b>	Mensagem que será exibida e atualizada na régua de processamento a cada execução da função IncRegua(), sendo que a taxa de atualização da interface é controlada pelo Binário.
------------------	--

**Retorno:**

<b>Nenhum</b>	<b>()</b>
---------------	-----------

**Exemplo:**

```
...
dbSelectArea("SA1")
dbGoTop()
SetRegua(LastRec())
While !Eof()
    IncRegua("Avaliando cliente:" + SA1->A1_COD)
    If Li > 60
```

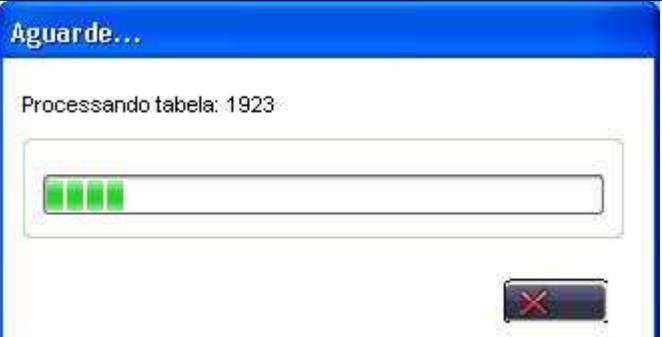
### **2.20.2 Processa()**

Régua de processamento simples, com apenas um indicador de progresso, utilizada no processamento de rotinas.

**Sintaxe:** **Processa(bAcao, cMensagem)**

**Retorno:** **Nil**

**Parâmetros:**

<b>bAcao</b>	Bloco de código que especifica a ação que será executada com o acompanhamento da régua de processamento.
<b>cMensagem</b>	Mensagem que será exibida na régua de processamento durante a execução.
<b>Aparência</b>	

**Exemplo: Função PROCESSA() e acessórios**

---

```
/*
+-----
| Função      | GPROCES1    | Autor | ROBSON LUIZ          | Data |
|             |             |       |                         |
+-----+
| Descrição      | Programa que demonstra a utilização das funções
PROCESSA()
|             | PROCREGUA() E INCPROC()
|             |
+-----+
| Uso          | Curso ADVPL
|             |
+-----+
*/
User Function GProces1()
Local aSay      := {}
Local aButton := {}
Local nOpc      := 0
Local cTitulo := "Exemplo de Funções"
Local cDesc1   := "Este programa exemplifica a utilização da função
Processa()"
Local cDesc2   := " em conjunto com as funções de incremento
ProcRegua() e"
```

```
Local cDesc3 := " IncProc()"
```

Exemplo (continuação):

```
Private cPerg := "PROCES"
```

```
CriaSX1()
```

```
Pergunte(cPerg,.F.)
```

```
AADD( aSay, cDesc1 )
```

```
AADD( aSay, cDesc2 )
```

```
AADD( aButton, { 5, .T., {|| Pergunte(cPerg,.T. ) } } )
```

```
AADD( aButton, { 1, .T., {|| nOpc := 1, FechaBatch() } } )
```

```
AADD( aButton, { 2, .T., {|| FechaBatch() } } )
```

```
FormBatch( cTitulo, aSay, aButton )
```

```
If nOpc <> 1
```

```
    Return Nil
```

```
Endif
```

```
Processa( {||End| RunProc(@|End)}, "Aguarde...", "Executando  
rotina.", .T. )
```

```
Return Nil
```

```
/*
```

```
+-----
```

```
-----
```

Função	RUNPROC	Autor   ROBSON LUIZ	Data
--------	---------	---------------------	------

```
|
```

```
+-----
```

```
-----
```

Descrição	Função de processamento executada através da PROCRESSA()
-----------	---

```
|
```

```
+-----
```

```
-----
```

Uso	Curso ADVPL
-----	-------------

```
|
```

```
+-----
```

```
-----
```

```
/*
```

```
Static Function RunProc(|End)
```

```
Local nCnt := 0
```

```
dbSelectArea("SX5")
```

```

dbSetOrder(1)
dbSeek(xFilial("SX5")+mv_par01,.T.)

dbEval( {|x| nCnt++
},,{| |X5_FILIAL==xFilial("SX5").And.X5_TABELA<=mv_par02|}

dbSeek(xFilial("SX5")+mv_par01,.T.)

ProcRegua(nCnt)
While !Eof() .And. X5_FILIAL == xFilial("SX5") .And. X5_TABELA <=
mv_par02
    IncProc("Processando tabela: "+SX5->X5_CHAVE)
    If IEnd
        MsgInfo(cCancela,"Fim")
        Exit
    Endif
    dbSkip()
End
Return .T.

```

## **SETPROC()**

---

A função SetProc() é utilizada para definir o valor máximo da régua de progressão criada através da função Processa().

**Sintaxe:** Processa(nMaxProc)

**Parâmetros:**

<b>nMaxProc</b>	Variável que indica o valor máximo de processamento (passos) que serão indicados pela régua.
-----------------	--

**Retorno:**

<b>Nenhum</b>	()
---------------	----

**Exemplo:**

```

...
dbSelectArea("SA1")
dbGoTop()
SetProc(LastRec())
While !Eof()
    IncProc()
    If Li > 60
...

```

## **INCPROC()**

---

A função IncProc() é utilizada para incrementar valor na régua de progressão criada através da função Processa()

**Sintaxe: IncProc(cMensagem)**

**Parâmetros:**

<b>cMensagem</b>	Mensagem que será exibida e atualizada na régua de processamento a cada execução da função IncProc(), sendo que a taxa de atualização da interface é controlada pelo Binário.
------------------	---

**Retorno:**

<b>Nenhum</b>	()
---------------	----

**Exemplo:**

```
...
dbSelectArea("SA1")
dbGoTop()
SetProc(LastRec())
While !Eof()
    IncProc("Avaliando cliente:" +SA1->A1_COD)
    If Li > 60
...
...
```

### **2.20.3 MsNewProcess().**

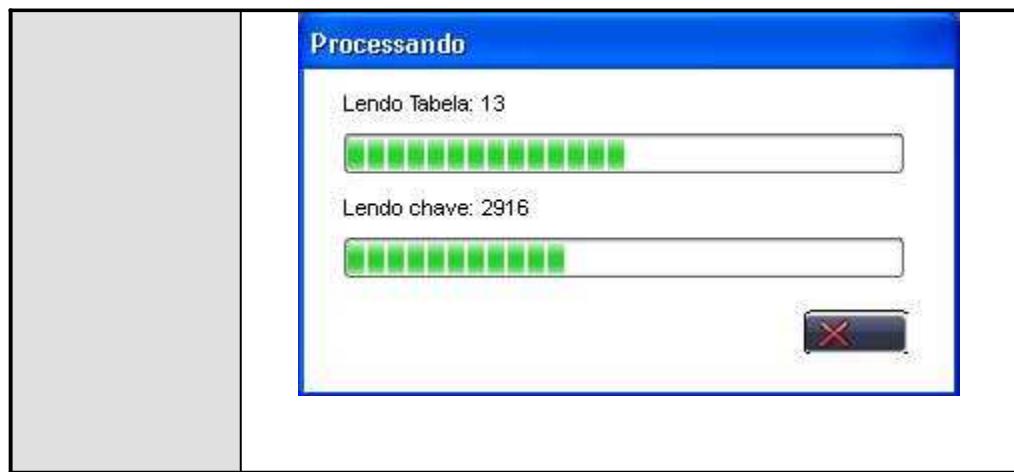
Régua de processamento dupla, possuindo dois indicadores de progresso independentes, utilizada no processamento de rotinas.

**Sintaxe: MsNewProcess():New(bAcao, cMensagem)**

**Retorno: oProcess ↗ objeto do tipo MsNewProcess()**

**Parâmetros:**

<b>bAcao</b>	Bloco de código que especifica a ação que será executada com o acompanhamento da régua de processamento.
<b>cMensagem</b>	Mensagem que será exibida na régua de processamento durante a execução.
<b>Aparência</b>	



## Métodos:

<b>Activate()</b>	Inicia a execução do objeto MsNewProcess instanciado.
<b>SetRegua1()</b>	Define a quantidade de informações que serão demonstradas pelo indicador de progresso superior. <b>Parâmetro:</b> nMaxProc
<b>IncRegua1()</b>	Incrementa em uma unidade o indicador de progresso superior, o qual irá demonstrar a evolução do processamento de acordo com a quantidade definida pelo método SetRegua1(). <b>Parâmetro:</b> cMensagem
<b>SetRegua2()</b>	Define a quantidade de informações que serão demonstradas pelo indicador de progresso inferior. <b>Parâmetro:</b> nMaxProc
<b>IncRegua2()</b>	Incrementa em uma unidade o indicador de progresso inferior, o qual irá demonstrar a evolução do processamento de acordo com a quantidade definida pelo método SetRegua2(). <b>Parâmetro:</b> cMensagem

## Exemplo: Objeto MsNewProcess() e métodos acessórios

---

```
/*
+-----
| Função      | GPROCES2    | Autor | ROBSON LUIZ           | Data |
|             |             |       |                         |
+-----+
| Descrição      | Programa que demonstra a utilização do objeto
MsNewProcess()   |
|                 | e seus métodos IncReguaX() e SetReguaX()
|             |             |
+-----+
| Uso          | Curso ADVPL
|             |
+-----+
*/
User Function GProces2()
Private oProcess := NIL

oProcess := MsNewProcess():New({|IEnd| RunProc(IEnd,oProcess)};
"Processando","Lendo...".T.)
oProcess:Activate()
```

```

Return Nil

/*
+-----
| Função      | RUNPROC      | Autor | ROBSON LUIZ           | Data |
|             |              |        |                         |
+-----+
| Descrição      | Função de processamento executada através da
MsNewProcess()   |
+-----+
| Uso          | Curso ADVPL
|             |              |
+-----+
*/
Static Function RunProc(lEnd,oObj)
Local i := 0
Local aTabela := {}
Local nCnt := 0

aTabela := {"00",0},{"13",0},{"35",0}, {"T3",0}

dbSelectArea("SX5")
cFilialSX5 := xFilial("SX5")
dbSetOrder(1)

For i:=1 To Len(aTabela)
    dbSeek(cFilialSX5+aTabela[i,1])
    While !Eof() .And. X5_FILIAL+X5_TABELA == cFilialSX5+aTabela[i,1]
        If lEnd
            Exit
        Endif
        nCnt++
        dbSkip()
    End
    aTabela[i,2] := nCnt
    nCnt := 0
Next i
oObj:SetRegua1(Len(aTabela))
For i:=1 To Len(aTabela)
    If lEnd
        Exit
    Endif
    oObj:IncRegua1("Lendo Tabela: "+aTabela[i,1])
    dbSelectArea("SX5")
    dbSeek(cFilialSX5+aTabela[i,1])
    oObj:SetRegua2(aTabela[i,2])

```

```

While !Eof() .And. X5_FILIAL+X5_TABELA == cFilialSX5+aTabela[i,1]
    oObj:IncRegua2("Lendo chave: "+X5_CHAVE)
    If !End
        Exit
    Endif
    dbSkip()
End
Next i
Return

```

#### 2.20.4 MsAguardar().

Indicador de processamento sem incremento.

**Sintaxe:** Processa(bAcao, cMensagem, cTitulo)

**Retorno:** Nil

**Parâmetros:**

<b>bAcao</b>	Bloco de código que especifica a ação que será executada com o acompanhamento da régua de processamento.
<b>cMensagem</b>	Mensagem que será exibida na régua de processamento durante a execução.
<b>cTitulo</b>	Título da janela da régua de processamento.
<b>Aparência</b>	

**Exemplo:** MSAGUARDE()

```

/*
+-----
| Função      | GMSAGUARDE | Autor | ROBSON LUIZ           | Data |
|             |            |       |                         |
+-----+
| Descrição   | Programa que demonstra a utilização das funções
MSAGUARDE()  |
|             |

```

```

|           | e MSProCTXT()
|
+-----+
| Uso      | Curso ADVPL
|
+-----+
|/|
-----+
USER FUNCTION GMsAguarda()
PRIVATE lEnd := .F.

MsAguarda({|lEnd| RunProc(@lEnd}),"Aguarde..." , "Processando
Clientes",.T.)

RETURN

/*
+-----+
| Função      | RUNPROC     | Autor | ROBSON LUIZ          | Data |
|
+-----+
| Descrição      | Função de processamento
|
+-----+
| Uso      | Curso ADVPL
|
+-----+
|/|
-----+
STATIC FUNCTION RunProc(lEnd)

dbSelectArea("SX5")
dbSetOrder(1)
dbGoTop()

While !Eof()
  If lEnd
    MsgInfo(cCancel,"Fim")
    Exit
  Endif
  MsProcTxt("Tabela: "+SX5->X5_TABELA+" Chave: "+SX5->X5_CHAVE)
  dbSkip()
End

RETURN

```

## 2.20.5MsgRun().

Indicador de processamento sem incremento.

**Sintaxe:** Processa(cMensagem, cTitulo, bAcao)

**Retorno:** Nil

### Parâmetros:

<b>cMensagem</b>	Mensagem que será exibida na régua de processamento durante a execução.
<b>bAcao</b>	Título da janela da régua de processamento. Bloco de código que especifica a ação que será executada com o acompanhamento da régua de processamento.
<b>Aparência</b>	<p style="background-color: #0070C0; color: white; padding: 2px;"><b>Título opcional</b></p> <p style="background-color: #F0F8FF; border: 1px solid #0070C0; padding: 10px; text-align: center;"><i>Lendo arquivo, aguarde...</i></p>

### Exemplo: MSGRun()

---

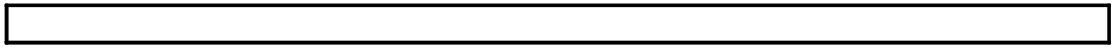
```
/*
+-----
| Função      | GMSGRUN      | Autor | ROBSON LUIZ           | Data |
|             |              |       |                         |
+-----+
| Descrição      | Programa que demonstra a utilização das funções
MSGRUN()
|             | e DBEVAL()
|             |
+-----+
| Uso          | Curso ADVPL
|             |
+-----+
*/
USER FUNCTION GMsgRun()
LOCAL nCnt := 0

dbSelectArea("SX1")
dbGoTop()

MsgRun("Lendo arquivo, aguarde...","Título opcional",{} | dbEval({|x|
nCnt++})})

MsgInfo("Ufa!!!, li "+AllTrim(Str(nCnt))+ " registros",FunName())

RETURN
```



## 2.21 ListBox()

A sintaxe clássica da linguagem ADVPL permite que o componente visual ListBox implemente dois tipos distintos de objetos:

**Lista simples:** lista de apenas uma coluna no formato de um vetor, a qual não necessita da especificação de um cabeçalho.

**Lista com colunas:** lista com diversas colunas que necessita de um cabeçalho no formato de um array de cabeçalho).

### 2.21.1 ListBox simples

**Sintaxe:**

```
@ nLinha,nColuna LISTBOX oListbox VAR nLista ITEMS aLista SIZE  
nLargura,nAltura OF oObjetoRef UNIDADE ON CHANGE CHANGE
```

**Parâmetros:**

<b>nLinha,nColuna</b>	Posição do objeto ListBox em função da janela em que ele será definido.
<b>oListBox</b>	Objeto ListBox que será criado.
<b>nLista</b>	Variável numérica que contém o número do item selecionado no ListBox.
<b>aLista</b>	Vetor simples contendo as strings que serão exibidas no ListBox.
<b>nLargura,nAltura</b>	Dimensões do objeto ListBox.
<b>oObjetoRef</b>	Objeto <i>dialog</i> no qual o componente será definido.
<b>UNIDADE</b>	Unidade de medida das dimensões: PIXEL.
<b>CHANGE</b>	Função ou lista de expressões que será executada na seleção de um item do ListBox.

### Aparência:



### Exemplo: LISTBOX como lista simples

```
#include "protheus.ch"

/*
+-----
| Função      | LISTBOXITE | Autor | ROBSON LUIZ           |Data |
|             |            |        |                      |
+-----+
| Descrição    | Programa que demonstra a utilização do LISTBOX()
como lista   | simples. |
|             |            |
+-----+
| Uso          | Curso ADVPL
|             |
+-----+
*/
User Function ListBoxIt()

Local aVetor  := {}
Local oDlg     := Nil
Local oLbx     := Nil
Local cTitulo  := "Consulta Tabela"
Local nChave   := 0
Local cChave   := ""

dbSelectArea("SX5")
```

```

dbSetOrder(1)
dbSeek(xFilial("SX5"))

CursorWait()

//+-----+
//| Carrega o vetor conforme a condição |
//+-----+
While !Eof() .And. X5_FILIAL == xFilial("SX5") .And. X5_TABELA=="00"
    AADD( aVetor, Trim(X5_CHAVE)+" - "+Capital(Trim(X5_DESCRI)) )
    dbSkip()
End

CursorArrow()

If Len( aVetor ) == 0
    Aviso( cTitulo, "Não existe dados a consultar", {"Ok"} )
    Return
Endif

//+-----+
//| Monta a tela para usuário visualizar consulta |
//+-----+
DEFINE MSDIALOG oDlg TITLE cTitulo FROM 0,0 TO 240,500 PIXEL
@ 10,10 LISTBOX oLbx VAR nChave ITEMS aVetor SIZE 230,95 OF oDlg
PIXEL
oLbx:bChange := {|| cChave := SubStr(aVetor[nChave],1,2) }
DEFINE SBUTTON FROM 107,183 TYPE 14 ACTION LoadTable(cChave) ENABLE
OF oDlg
DEFINE SBUTTON FROM 107,213 TYPE 1 ACTION oDlg:End() ENABLE OF oDlg

ACTIVATE MSDIALOG oDlg CENTER

Return

```

**Exemplo: LISTBOX como lista simples – funções acessórias**

```

*/
+-----+
----+
| Função      | LISTBOXITE | Autor | ROBSON LUIZ           | Data |
|             |            |        |                      |
+-----+
----+
| Descrição   | Função que carrega os dados da tabela selecionada
em um          |           |
|             | listbox. |
|             |

```

```

+-----+
| Uso      | Curso ADVPL
|
+-----+
| */
|
STATIC FUNCTION LoadTable(cTabela)

LOCAL aTabela := {}
LOCAL oDlg := NIL
LOCAL oLbx := NIL

dbSelectArea("SX5")
dbSeek(xFilial("SX5") + cTabela)

//+
//| O vetor pode receber carga de duas maneiras, acompanhe... |
//+
//| Utilizando While/End
//+-----+

dbEval({|| AADD(aTabela,{X5_CHAVE,Capital(X5_DESCRI)})},,{|| 
X5_TABELA==cTabela})

If Len(aTabela)==0
    Aviso( "FIM", "Necessário selecionar um item", {"Ok"} )
    Return
Endif

DEFINE MSDIALOG oDlg TITLE "Dados da tabela selecionada" FROM
300,400 TO 540,900 PIXEL
@ 10,10 LISTBOX oLbx FIELDS HEADER "Tabela", "Descrição" SIZE
230,095 OF oDlg PIXEL
oLbx:SetArray( aTabela )
oLbx:bLine := {|| {aTabela[oLbx:nAt,1],aTabela[oLbx:nAt,2]} }

DEFINE SBUTTON FROM 107,213 TYPE 1 ACTION oDlg:End() ENABLE OF oDlg
ACTIVATE MSDIALOG oDlg

RETURN

```

## 2.21.2 ListBox múltiplas colunas

### Sintaxe:

```
@ nLinha,nColuna LISTBOX oListbox FIELDS HEADER "Header1", ..., "HeaderX"  
SIZE nLargura,nAltura OF oObjetoRef UNIDADE
```

**Parâmetros:**

<b>nLinha,nColuna</b>	Posição do objeto ListBox em função da janela em que ele será definido.
<b>oListBox</b>	Objeto ListBox que será criado.
<b>nLista</b>	Variável numérica que contém o número do item selecionado no ListBox.
<b>“Header1”,...,“HeaderX”</b>	Strings identificando os títulos das colunas do Grid.
<b>nLargura,nAltura</b>	Dimensões do objeto ListBox.
<b>oObjetoRef</b>	Objeto <i>dialog</i> no qual o componente será definido.
<b>UNIDADE</b>	Unidade de medida das dimensões: PIXEL.
<b>CHANGE</b>	Função ou lista de expressões que será executada na seleção de um item do ListBox.

**Métodos:**

<b>SetArray()</b>	Método o objeto ListBox que define qual <i>array</i> contém os dados que serão exibidos no <i>grid</i> .
-------------------	--

**Atributos:**

<b>bLine</b>	Atributo do objeto ListBox que vincula cada linha,coluna do <i>array</i> , com cada cabeçalho do <i>grid</i> .
--------------	--

**Aparência:**



## Exemplo: LISTBOX com grid

---

```
#include "protheus.ch"

/*
+-----
| Função      | LIST_BOX   | Autor | ROBSON LUIZ           | Data |
|             |            |       |                         |
+-----+
| Descrição      | Programa que demonstra a utilização de um
LISTBOX() com|
|                 | grid.|
|             |            |
+-----+
| Uso      | Curso ADVPL
|             |
+-----+
*/
User Function List_Box()

Local aVetor := {}
Local oDlg
Local oLbx
Local cTitulo := "Cadastro de Bancos"
Local cFilSA6

dbSelectArea("SA6")
dbSetOrder(1)
cFilSA6 := xFilial("SA6")
dbSeek(cFilSA6)

// Carrega o vetor conforme a condição.
While !Eof() .And. A6_FILIAL == cFilSA6
    AADD( aVetor, { A6_COD, A6_AGENCIA, A6_NUMCON, A6_NOME,
A6_NREDUZ, A6_BAIRRO, A6_MUN } )
    dbSkip()
End

// Se não houver dados no vetor, avisar usuário e abandonar rotina.
If Len( aVetor ) == 0
    Aviso( cTitulo, "Não existe dados a consultar", {"Ok"} )
    Return
Endif
```

```

// Monta a tela para usuário visualizar consulta.
DEFINE MSDIALOG oDlg TITLE cTitulo FROM 0,0 TO 240,500 PIXEL

    // Primeira opção para montar o listbox.
    @ 10,10 LISTBOX oLbx FIELDS HEADER ;
    "Banco", "Agencia", "C/C", "Nome Banco", "Fantasia", "Bairro",
    "Município" ;
    SIZE 230,95 OF oDlg PIXEL

    oLbx:SetArray( aVetor )
    oLbx:bLine := {|| {aVetor[oLbx:nAt,1];
                        aVetor[oLbx:nAt,2];
                        aVetor[oLbx:nAt,3];
                        aVetor[oLbx:nAt,4];
                        aVetor[oLbx:nAt,5];
                        aVetor[oLbx:nAt,6];
                        aVetor[oLbx:nAt,7]}}

```

Exemplo (continuação):

```

// Segunda opção para monta o listbox
/*
oLbx :=
TWBrowse():New(10,10,230,95,,aCabecalho,,oDlg,,,,,,,,,,F,,,T,,,,
F,,,)
oLbx:SetArray( aVetor )
oLbx:bLine := {|| aEval(aVetor[oLbx:nAt],{|z,w|
aVetor[oLbx:nAt,w] }) }
*/

```

DEFINE SBUTTON FROM 107,213 TYPE 1 ACTION oDlg:End() ENABLE OF oDlg  
ACTIVATE MSDIALOG oDlg CENTER

Return

## 2.22 ScrollBox()

O ScrollBox é o objeto utilizado para permitir que um *dialog* exiba barras de rolagem verticais e Horizontais. Algumas aplicações com objetos definem automaticamente o ScrollBox, tais como:

- . Enchoice() ou MsMGet()
- . NewGetDados()
- . ListBox()

Quando é definido um objeto ScrollBox, os demais componentes da janela deverão referenciar este objeto e não mais o objeto *dialog*.

Desta forma o ScrollBox é atribuído a um objeto ScrollBox. *dialog*, e os componentes ao

- ¶ MsDialog() ¶ ScrollBox()
- ¶ ScrollBox() ¶ Componentes Visuais

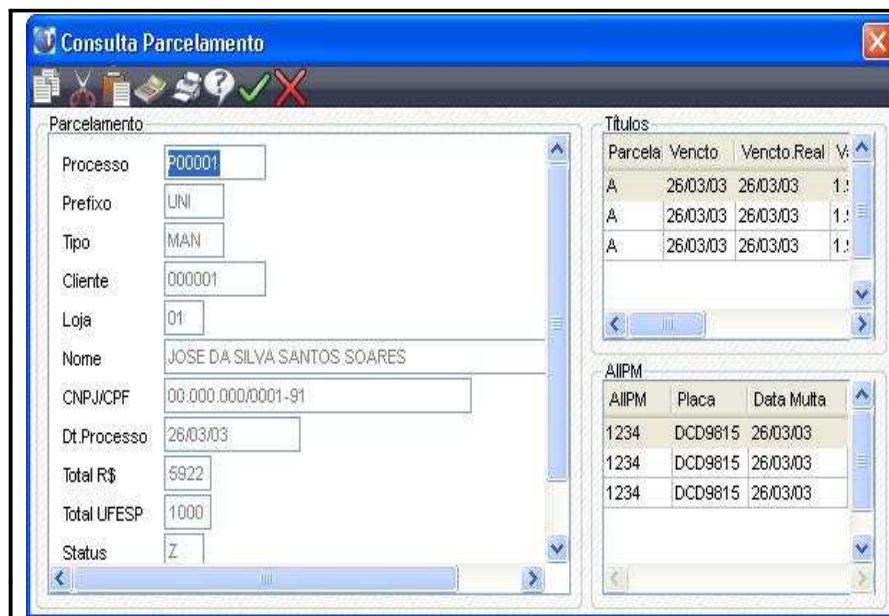
**Sintaxe:**

```
@ nLinha,nColuna SCROLLBOX oScrollView HORIZONTAL VERTICAL SIZE  
nLargura,nAltura OF oObjetoRef BORDER
```

**Parâmetros:**

<b>nLinha,nColuna</b>	Posição do objeto ScrollBox em função da janela em que ele será definido.
<b>oScrollView</b>	Objeto ScrollBox que será criado.
<b>HORIZONTAL</b>	Parâmetro que quando definido habilita a régua de rolagem horizontal.
<b>VERTICAL</b>	Parâmetro que quando definido habilita a régua de rolagem vertical.
<b>nLargura,nAltura</b>	Dimensões do objeto ScrollBox.
<b>oObjetoRef</b>	Objeto dialog no qual o componente será definido.
<b>BORDER</b>	Parâmetro que quando definido habilita a exibição de uma borda de delimitação do ScrollBox em relação a outros objetos.

**Aparência:**



## Exemplo: Utilização de múltiplos ScrollBoxes

---

```
#INCLUDE "PROTHEUS.CH"

/*
+-----
| Função      | SCROLL()      | Autor | ROBSON LUIZ      | Data |
|             |               |       |                   |
+-----+
| Descrição      | Programa que demonstra como montar uma enchoice
apenas |               | com variáveis, incluindo o recurso de rolagem.
|             |               |
+-----+
| Uso      | Curso ADVPL
|             |
+-----+
|             |
+-----+
*/
USER FUNCTION Scroll()

LOCAL oDlg := NIL
LOCAL oScroll := NIL
LOCAL oLbx1 := NIL
LOCAL oLbx2 := NIL
LOCAL bGet := NIL
LOCAL oGet := NIL
LOCAL aAIIPM := {}
LOCAL aTitulo := {}
LOCAL nTop := 5
LOCAL nWidth := 0
LOCAL cGet := ""
LOCAL cPict := ""
LOCAL cVar := ""
LOCAL n := 0

PRIVATE cTitulo := "Consulta Parcelamento"
PRIVATE aSay := {}
PRIVATE cProcesso,cPrefixo,cTipo,cCliente,cLoja,cNome,cCGC
PRIVATE dData,nTotal,nUFESP,cStatus,cCond

cProcesso := "P00001"
cPrefixo := "UNI"
cTipo := "MAN"
cCliente := "000001"
```

```

cLoja      := "01"
cNome       := "JOSE DA SILVA SANTOS SOARES"
cCGC        := "00.000.000/0001-91"
dData       := "26/03/03"
nTotal      := 5922.00
nUFESP     := 1000.00
cStatus     := "Z"
cCond       := "001"

// Vetor para os campos no Scroll Box
//+-----+
//| aSay[n][1] - Titulo          |
//| aSay[n][2] - Tipo            |
//| aSay[n][3] - Tamanho          |
//| aSay[n][4] - Decimal          |
//| aSay[n][5] - Conteúdo/Variável |
//| aSay[n][6] - Formato          |
//+-----+
AADD(aSay,{"Processo"      , "C",06,0,"cProcesso" , "@!"})
AADD(aSay,{"Prefixo"       , "C",03,0,"cPrefixo"   , "@!"})
AADD(aSay,{"Tipo"          , "C",03,0,"cTipo"      , "@!"})
AADD(aSay,{"Cliente"      , "C",06,0,"cCliente"   , "@!"})
AADD(aSay,{"Loja"          , "C",02,0,"cLoja"      , "@!"})
AADD(aSay,{"Nome"          , "C",30,0,"cNome"      , "@!"})
AADD(aSay,{"CNPJ/CPF"     , "C",14,0,"cCGC"       , "@!"})
AADD(aSay,{"Dt.Processo"  , "D",08,0,"dData"      , "@!"})
AADD(aSay,{"Total R$"    , "N",17,2,"nTotal"     , "@!"})
AADD(aSay,{"Total UFESP" , "N",17,2,"nUFESP"    , "@!"})
AADD(aSay,{"Status"       , "C",01,0,"cStatus"    , "@!"})
AADD(aSay,{"Cond.Pagto"   , "C",03,0,"cCond"      , "@!"})

// Vetor para List Box
AADD(aAIIPM,{"1234","DCD9815","26/03/03"})
AADD(aAIIPM,{"1234","DCD9815","26/03/03"})
AADD(aAIIPM,{"1234","DCD9815","26/03/03"})

// Vetor para List Box
AADD(aTitulo,{"A","26/03/03","26/03/03","1.974,00","100,00"})
AADD(aTitulo,{"A","26/03/03","26/03/03","1.974,00","100,00"})
AADD(aTitulo,{"A","26/03/03","26/03/03","1.974,00","100,00"})

DEFINE MSDIALOG oDlg TITLE cTitulo FROM 122,0 TO 432,600 OF oDlg
PIXEL
@ 013,002 TO 154,192 LABEL "Parcelamento"  OF oDlg PIXEL
@ 013,195 TO 082,298 LABEL "Títulos"        OF oDlg PIXEL
@ 083,195 TO 154,298 LABEL "AIIPM"           OF oDlg PIXEL

//scrollbox

```

```

@ 019,006 SCROLLBOX oScroll HORIZONTAL VERTICAL SIZE 131,182 OF
oDlg BORDER
For n:=1 TO Len(aSay)

    bGet := &("{| | "+aSay[n][1]+"}")
    cVar := aSay[n][5]
    cGet := "| u | IIF(PCount()>0,"+cVar+":=u,"+cVar+")"
    cPict := aSay[n][6]

    TSay():New(nTop,5,bGet,oScroll,,,F.,,F.,,T.,,,;
GetTextWidth(0,Trim(aSay[n][1])),15,;
.F.,,F.,,F.,,F.)
    oGet:=TGet():New(nTop-
2,40,&cGet,oScroll,,7,cPict,,,,,F.,,T.,;;
.,F.,,F.,,T.,,F.,,(cVar),,,,T.)
    nTop+=11
Next n

//listbox títulos
@ 019,199 LISTBOX oLbx1 FIELDS HEADER ;
    "Parcela","Vencto","Vencto.Real","Valor R$","Qtd.UFESP";
    COLSIZES 21,24,33,63,100;
    SIZE 095,059 OF oDlg PIXEL
    oLbx1:SetArray( aTitulo )
    oLbx1:bLine := {| |{aTitulo[oLbx1:nAt,1],aTitulo[oLbx1:nAt,2],;
aTitulo[oLbx1:nAt,3],aTitulo[oLbx1:nAt,4],aTitulo[oLbx1:nAt,5]}}

//listbox aiipm
@ 089,199 LISTBOX oLbx2 FIELDS HEADER "AIIPM","Placa","Data
Multas" ;
    COLSIZES 24,21,30 SIZE 095,061 OF oDlg PIXEL
    oLbx2:SetArray( aAIIPM )
    oLbx2:bLine :=
{| |{aAIIPM[oLbx2:nAt,1],aAIIPM[oLbx2:nAt,2],aAIIPM[oLbx2:nAt,3]}}

ACTIVATE MSDIALOG oDlg CENTER ON INIT
EnchoiceBar(oDlg,{| |oDlg:End()},{| |oDlg:End()})

RETURN

```

## 2.23 ParamBox()

Implementa uma tela de parâmetros, que não necessita da criação de um grupo de perguntas no SX1, e com funcionalidades que a Pergunte() não disponibiliza, tais como CheckBox e RadioButtons.

Cada componente da ParamBox será associado a um parâmetro Private denominado MV\_PARxx, de acordo com a ordem do componente na tela. Os parâmetros da

ParamBox podem ser utilizados de forma independente em uma rotina específica, ou complementando opções de uma rotina padrão.

## Cuidados

---

- A PARAMBOX define os parâmetros seguindo o princípio das variáveis MV\_PARxx. Caso ela seja utilizada em uma rotina em conjunto com parâmetros padrões (SX1 + Pergunte()) é necessário salvar os parâmetros padrões, chamar a Parambox(), salvar o retorno da Parambox() em variáveis Private específicas (MVPARBOXxx) e depois restaurar os parâmetros padrões, conforme o exemplo desta documentação.
- O objeto COMBO() da PARAMBOX() possui um problema em seu retorno: Caso o combo não seja selecionado, ele manterá seu conteúdo como numérico, caso seja, ele receberá o texto da opção e não o número da opção. O exemplo desta documentação ilustra o tratamento de código necessário para proteger a aplicação.
- Ao utilizar a ParamBox em uma função que também utilize parâmetros definidos pela função Pergunte() deve-se:
  - Salvar e restaurar os MV\_PARs da Pergunte()
  - Definir variáveis Private próprias para a ParamBox, as quais irão armazenar o conteúdo das MV\_PARs que esta retorna.

**Sintaxe:** ParamBox (aParamBox, cTitulo, aRet, bOk, aButtons, lCentered,; nPosx, nPosy, oMainDlg, cLoad, lCanSave, lUserSave)

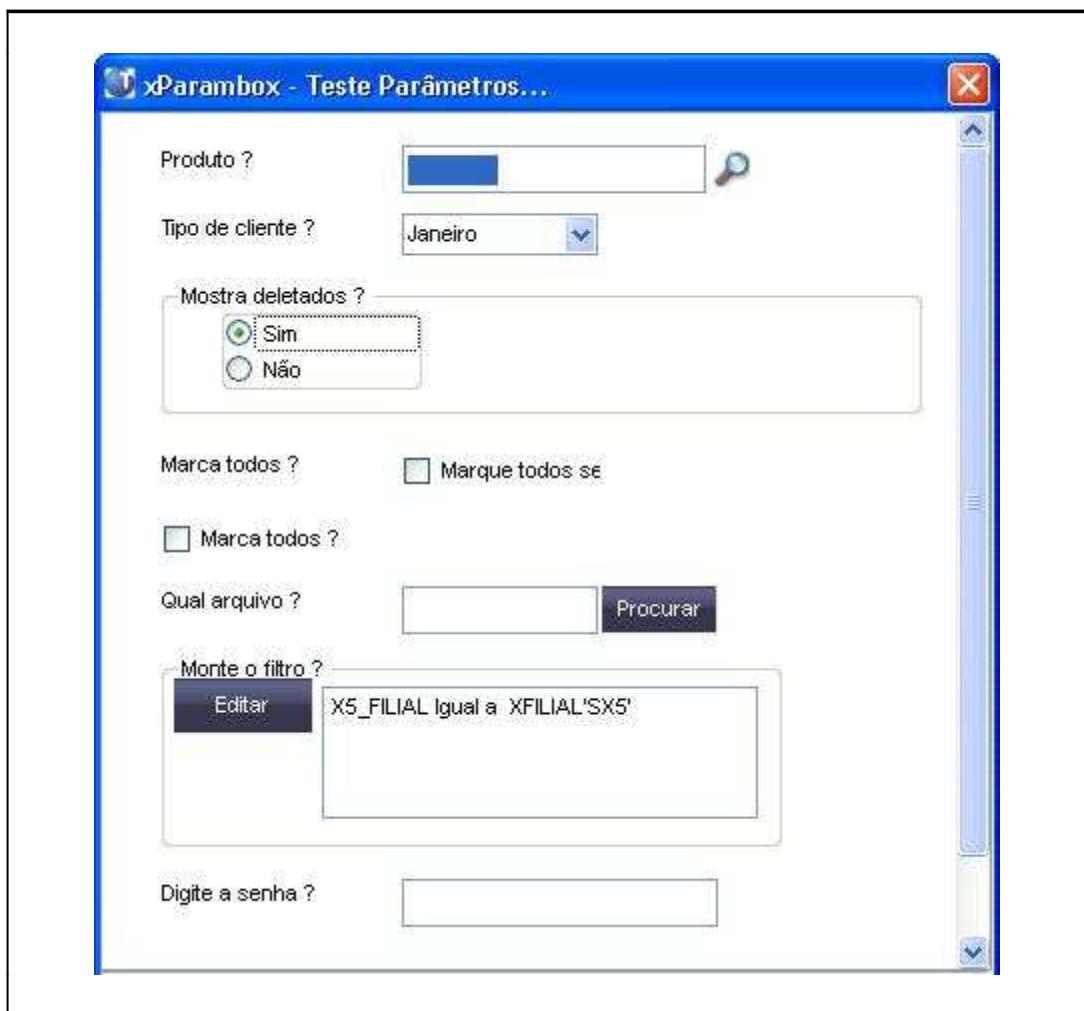
**Retorno:** lOK ■ indica se a tela de parâmetros foi cancelada ou confirmada

**Parâmetros:**

<b>aParamBox</b>	Array de parâmetros de acordo com a regra da ParamBox.
<b>cTitulo</b>	Titulo da janela de parâmetros.
<b>aRet</b>	Array que será passado por referência e retornado com o conteúdo de cada parâmetro.
<b>bOk</b>	Bloco de código para validação do OK da tela de parâmetros
<b>aButtons</b>	Array contendo a regra para adição de novos botões (além do OK e Cancelar) // AADD(aButtons,{nType,bAction,cTexto})
<b>lCentered</b>	Se a tela será exibida centralizada, quando a mesma não estiver vinculada a outra janela.
<b>nPosx</b>	Posição inicial -> linha (Linha final: nPosX+274).
<b>nPosy</b>	Posição inicial -> coluna (Coluna final: nPosY+445).
<b>oMainDlg</b>	Caso o ParamBox deva ser vinculado a uma outra tela.

<b>cLoad</b>	Nome do arquivo aonde as respostas do usuário serão salvas / lidas.
<b>ICanSave</b>	Se as respostas para as perguntas podem ser salvas.
<b>IUserSave</b>	Se o usuário pode salvar sua própria configuração.

Aparência:



**Regras do array aParamBox:**

**[1] Tipo do parâmetro:** Para cada tipo de parâmetro as demais posições do array variam de conteúdo conforme abaixo:

**1 - MsGet**

- [2] : Descrição
- [3] : String contendo o inicializador do campo
- [4] : String contendo a Picture do campo
- [5] : String contendo a validação
- [6] : Consulta F3
- [7] : String contendo a validação When
- [8] : Tamanho do MsGet
- [9] : Flag .T./.F. Parâmetro Obrigatório?

**2 - Combo**

- [2] : Descrição
- [3] : Numérico contendo a opção inicial do combo
- [4] : Array contendo as opções do Combo
- [5] : Tamanho do Combo

[6] : Validação

[7] : Flag .T./.F. Parâmetro Obrigatório?

### **3 - Radio**

[2] : Descrição

[3] : Numérico contendo a opção inicial do Rádio

[4] : Array contendo as opções do Rádio

[5] : Tamanho do Rádio

[6] : Validação

[7] : Flag .T./.F. Parâmetro Obrigatório?

### **4 - CheckBox ( Com Say )**

[2] : Descrição

[3] : Indicador Lógico contendo o inicial do Check

[4] : Texto do CheckBox

[5] : Tamanho do Rádio

[6] : Validação

[7] : Flag .T./.F. Parâmetro Obrigatório?

### **5 - CheckBox ( linha inteira )**

[2] : Descrição

[3] : Indicador Lógico contendo o inicial do Check

[4] : Tamanho do Rádio

[5] : Validação

[6] : Flag .T./.F. Parâmetro Obrigatório?

### **6 - File**

[2] : Descrição

[3] : String contendo o inicializador do campo

[4] : String contendo a *Picture* do campo

[5] : String contendo a validação

[6] : String contendo a validação *When*

[7] : Tamanho do MsGet

[8] : Flag .T./.F. Parâmetro Obrigatório ?

[9] : Texto contendo os tipos de arquivo

Ex.: "Arquivos .CSV | \*.CSV"

[10]: Diretório inicial do CGETFILE()

[11]: Parâmetros do CGETFILE()

### **7 - Montagem de expressão de filtro**

[2] : Descrição

[3] : Alias da tabela

[4] : Filtro inicial

[5] : Opcional - Cláusula *When* Botão Editar Filtro

### **8 - MsGet Password**

[2] : Descrição

- [3] : String contendo o inicializador do campo
- [4] : String contendo a Picture do campo
- [5] : String contendo a validação
- [6] : Consulta F3
- [7] : String contendo a validação When
- [8] : Tamanho do MsGet
- [9] : Flag .T./.F. Parâmetro Obrigatório?

#### **9 - MsGet Say**

- [2] : String Contendo o Texto a ser apresentado
- [3] : Tamanho da String
- [4] : Altura da String
- [5] : Negrito (lógico)

#### **Exemplo: Utilização da ParamBox()**

---

```
#include "protheus.ch"

/*
+-----
| Função      | xParamBox    | Autor | ROBSON LUIZ           | Data |
|             |              |       |                         |
+-----+
| Descrição   | Programa que demonstra a utilização da PARAMBOX
como       |
|             | forma alternativa de disponibilizar parâmetros em um
|             | processamento.
|             |
+-----+
| Uso        | Curso ADVPL
|             |
+-----+
*/
User Function xParamBox()

Local aRet := {}
Local aParamBox := {}
Local aCombo :=
{"Janeiro","Fevereiro","Março","Abril","Maio","Junho","Julho","Agosto","Setembro","Outubro","Novembro","Dezembro"}
Local i      := 0
Private cCadastro := "xParambox"
```

```

AADD(aParamBox,{1,"Produto",Space(15),"""","SB1","",0,.F.})
AADD(aParamBox,{2,"Tipo de cliente",1,aCombo,50,"",.F.})

AADD(aParamBox,{3,"Mostra
deletados",IIF(Set(_SET_DELETED),1,2),{"Sim","Não"},50,"",.F.})

AADD(aParamBox,{4,"Marca todos ?",.F.,"Marque todos se necessário
for.",50,"",.F.})

AADD(aParamBox,{5,"Marca todos ?",.F.,50,"",.F.})

AADD(aParamBox,{6,"Qual arquivo",Space(50),"""","",50,.F.;;
"Arquivo .DBF | *.DBF"})

AADD(aParamBox,{7,"Monte o
filtro","SX5","X5_FILIAL==xFilial('SX5')"})
AADD(aParamBox,{8,"Digite a senha",Space(15),"""","",80,.F.})

If ParamBox(aParamBox,"Teste Parâmetros...",@"aRet)
    For i:=1 To Len(aRet)
        MsgInfo(aRet[i],"Opção escolhida")
    Next
Endif

Return

```

#### **Exemplo: Protegendo os parâmetros MV\_PARs da Pergunte() em uso.**

```

#include "protheus.ch"

/*
+-----
| Função      | XPARBOX()   | Autor | ARNALDO RAYMUNDO JR. | Data |
|             |             |       |
+-----
| Descrição      | Função utilizando a PARAMBOX() e protegendo os
MV_PARs|
|             | ativos do programa principal.
|             |
+-----
| Uso          | Curso ADVPL
|             |
+-----
*/
Static Function XPARBOX(cPerg)

```

```

Local aParamBox := {}
Local cTitulo := "Transferência para Operação"
Local bOk := {|| .T.}
Local aButtons := {}; Local aRet := {}
Local nPosx; Local nPosy; Local nX := 0
Local cLoad := ""
Local lCentered := .T.; Local lCanSave := .F.; Local lUserSave := .F.
Local aParamAtu := Array(4)

// Salva as perguntas padrões antes da chamada da ParamBox
For nX := 1 to Len(aParamAtu)
    aParamAtu [nX] := &("Mv_Par"+StrZero(nX,2))
Next nX

AADD(aParamBox,{2,"Atualiza taxa de depreciação?", 2, {"Sim","Não"},100,;
        "AllwaysTrue()",.T.})

ParamBox(aParamBox, cTitulo, aRet, bOk, aButtons, lCentered, nPosx,
nPosy, /*oMainDlg*/ ,;
        cLoad, lCanSave, lUserSave)

IF ValType(aRet) == "A" .AND. Len(aRet) == Len(aParamBox)
    For nX := 1 to Len(aParamBox)
        If aParamBox[nX][1] == 1
            &("MvParBox"+StrZero(nX,2)) := aRet[nX]
        Elseif aParamBox[nX][1] == 2 .AND. ValType(aRet[nX]) ==
"C"
            &("MvParBox"+StrZero(nX,2)) :=
aScan(aParamBox[nX][4],;
                {|x| Alltrim(x) == aRet[nX]})}
        Elseif aParamBox[nX][1] == 2 .AND. ValType(aRet[nX]) ==
"N"
            &("MvParBox"+StrZero(nX,2)) := aRet[nX]
        Endif
    Next nX
ENDIF

// Restaura as perguntas padrões apos a chamada da ParamBox
For nX := 1 to Len(aParamAtu)
    &("Mv_Par"+StrZero(nX,2)) := aParamAtu[nX]
Next nX

Return

```

## MÓDULO 05: INTRODUÇÃO A ORIENTAÇÃO À OBJETOS

### 12. Conceitos de orientação à objetos

O termo orientação a objetos pressupõe uma organização de software em termos de coleção de objetos discretos incorporando estrutura e comportamento próprios. Esta abordagem de organização é essencialmente diferente do desenvolvimento tradicional de software, onde estruturas de dados e rotinas são desenvolvidas de forma apenas fracamente acopladas.

Neste tópico serão os conceitos de programação orientada a objetos listados abaixo. Esta breve visão geral do paradigma permitirá entender melhor os conceitos associados à programação orientada a objetos e, em particular, às construções implementadas através da linguagem ADVPL.

- **Objetos**
- **Herança**
- **Atributos**
- **Métodos**
- **Classes**
- **Abstração**
- **Generalização**
- **Encapsulamento**
- **Polimorfismo**

### 2.24 Definições

#### **Objeto**

Um objeto é uma entidade do mundo real que tem uma identidade. Objetos podem representar entidades concretas (um arquivo no meu computador, uma bicicleta) ou entidades conceituais (uma estratégia de jogo, uma política de escalonamento em um sistema operacional). Cada objeto ter sua identidade significa que dois objetos são distintos mesmo que eles apresentem exatamente as mesmas características.

Embora objetos tenham existência própria no mundo real, em termos de linguagem de programação, um objeto necessita um mecanismo de identificação. Esta identificação de objeto deve ser única, uniforme e independente do conteúdo do objeto.

Este é um dos mecanismos que permite a criação de coleções de objetos, as quais são também objetos em si.

A estrutura de um objeto é representada em termos de atributos. O comportamento de um objeto é representado pelo conjunto de operações que podem ser executadas sobre o objeto.

## Classe

---

Objetos com a mesma estrutura e o mesmo comportamento são agrupados em classes. Uma classe é uma abstração que descreve propriedades importantes para uma aplicação e simplesmente ignora o resto.

Cada classe descreve um conjunto (possivelmente infinito) de objetos individuais. Cada objeto é dito ser uma instância de uma classe. Assim, cada instância de uma classe tem seus próprios valores para cada atributo, mas dividem os nomes dos atributos e métodos com as outras instâncias da classe. Implicitamente, cada objeto contém uma referência para sua própria classe, em outras palavras: ele sabe o que ele é.

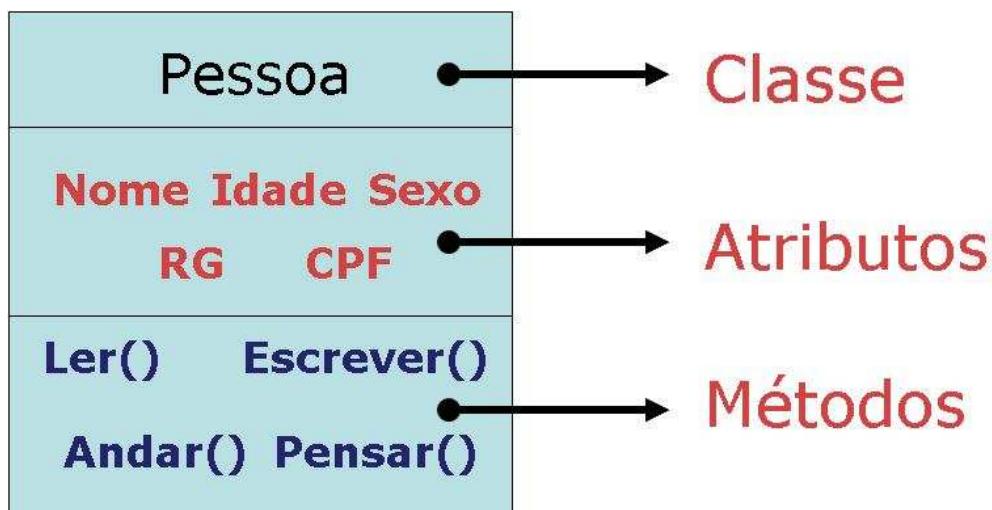
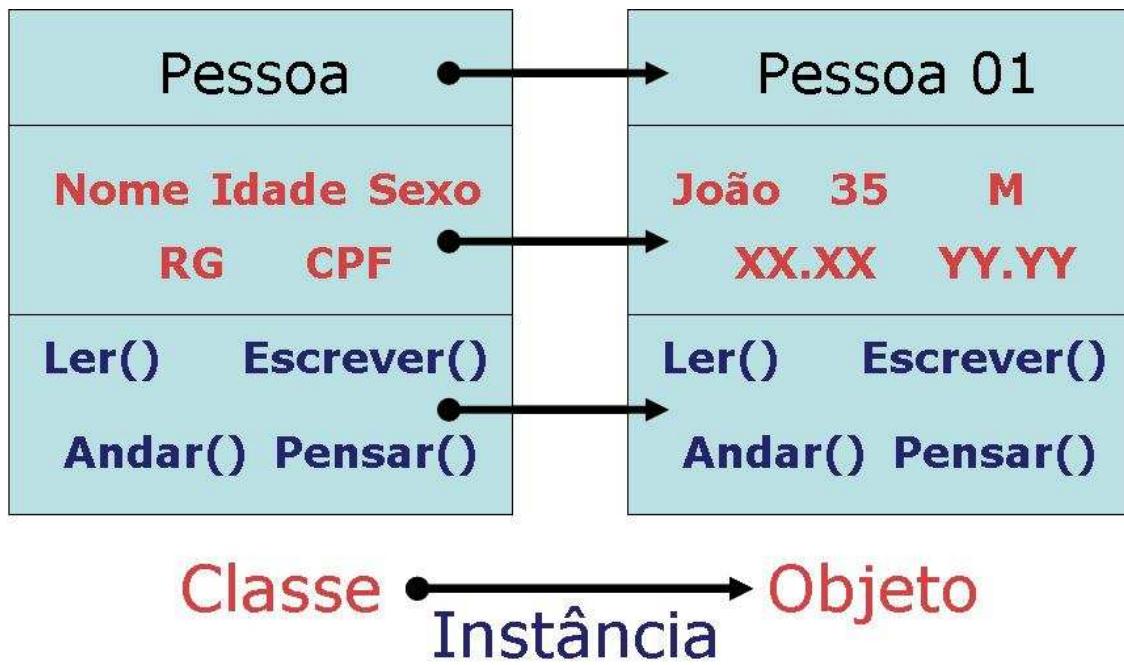


Figura: Representação de uma classe de objetos

---



**Figura:** Representação de um objeto instanciado de uma classe

---

## Polimorfismo

Polimorfismo significa que a mesma operação pode se comportar de forma diferente em classes diferentes. Por exemplo, a operação move quando aplicada a uma janela de um sistema de interfaces tem um comportamento distinto do que quando aplicada a uma peça de um jogo de xadrez. Um método é uma implementação específica de uma operação para uma certa classe.

Polimorfismo também implica que uma operação de uma mesma classe pode ser implementada por mais de um método. O usuário não precisa saber quantas implementações existem para uma operação, ou explicitar qual método deve ser utilizado: a linguagem de programação deve ser capaz de selecionar o método correto a partir do nome da operação, classe do objeto e argumentos para a operação. Desta forma, novas classes podem ser adicionadas sem necessidade de modificação de código já existente, pois cada classe apenas define os seus métodos e atributos.

No mundo real, alguns objetos e classes podem ser descritos como casos especiais, ou especializações, de outros objetos e classes. Por exemplo, a classe de computadores pessoais com processador da linha 80x86 é uma especialização de computadores pessoais, que por sua vez é uma especialização de computadores. Não é desejável que tudo que já foi descrito para computadores tenha de ser repetido para computadores pessoais ou para computadores pessoais com processador da linha 80x86.

### **Herança**

---

Herança é o mecanismo do paradigma de orientação a objetos que permite compartilhar atributos e operações entre classes baseada em um relacionamento hierárquico. Uma classe pode ser definida de forma genérica e depois refinada sucessivamente em termos de subclasses ou classes derivadas. Cada subclass incorpora, or herda, todas as propriedades de sua superclasse (ou classe base) e adiciona suas propriedades únicas e particulares. As propriedades da classe base não precisam ser repetidas em cada classe derivada. Esta capacidade de fatorar as propriedades comuns de diversas classes em uma superclasse pode reduzir significativamente a repetição de código em um projeto ou programa, sendo uma das principais vantagens da abordagem de orientação a objetos.

## **2.25 Conceitos Básicos**

A abordagem de orientação a objetos favorece a aplicação de diversos conceitos considerados fundamentais para o desenvolvimento de bons programas, tais como abstração e encapsulamento.

Tais conceitos não são exclusivos desta abordagem, mas são suportados de forma melhor no desenvolvimento orientado a objetos do que em outras metodologias.

---

### **Abstração**

Abstração consiste de focalizar nos aspectos essenciais inerentes a uma entidade e ignorar propriedades “acidentais”. Em termos de desenvolvimento de sistemas, isto significa concentrar-se no que um objeto é e faz antes de se decidir como ele será implementado. O uso de abstração preserva a liberdade para tomar decisões de desenvolvimento ou de implementação apenas quando há um melhor entendimento do problema a ser resolvido.

Muitas linguagens de programação modernas suportam o conceito de abstração de dados; porém, o uso de abstração juntamente com polimorfismo e herança, como suportado em orientação a objetos, é um mecanismo muito mais poderoso.

O uso apropriado de abstração permite que um mesmo modelo conceitual (orientação a objetos) seja utilizado para todas as fases de desenvolvimento de um sistema, desde sua análise até sua documentação.

## **Encapsulamento**

---

Encapsulamento, também referido como esconder informação, consiste em separar os aspectos externos de um objeto, os quais são acessíveis a outros objetos, dos detalhes internos de implementação do objeto, os quais permanecem escondidos dos outros objetos. O uso de encapsulamento evita que um programa torne-se tão interdependente que uma pequena mudança tenha grandes efeitos colaterais.

O uso de encapsulamento permite que a implementação de um objeto possa ser modificada sem afetar as aplicações que usam este objeto. Motivos para modificar a implementação de um objeto podem ser, por exemplo, melhoria de desempenho, correção de erros e mudança de plataforma de execução.

Assim como abstração, o conceito de Encapsulamento não é exclusivo da abordagem de orientação a objetos. Entretanto, a habilidade de se combinar estrutura de dados e comportamento em uma única entidade torna o Encapsulamento mais elegante e mais poderoso do que em linguagens convencionais que separam estruturas de dados e comportamento.

## **Compartilhamento**

---

Técnicas de orientação a objetos promovem compartilhamento em diversos níveis distintos. Herança de estrutura de dados e comportamento permite que estruturas comuns sejam compartilhadas entre diversas classes derivadas similares sem redundância. O compartilhamento de código usando herança é uma das grandes vantagens da orientação a objetos. Ainda mais importante que a economia de código é a clareza conceitual de reconhecer que operações diferentes são na verdade a mesma coisa, o que reduz o número de casos distintos que devem ser entendidos e analisados.

O desenvolvimento orientado a objetos não apenas permite que a informação dentro de um projeto seja compartilhada como também oferece a possibilidade de reutilizar projetos e código em projetos futuros. As ferramentas para alcançar este compartilhamento, tais como abstração, Encapsulamento e herança, estão presentes na metodologia; uma estratégia de reuso entre projetos é a definição de bibliotecas de elementos reusáveis. Entretanto, orientação a objetos não é uma fórmula mágica para alcançar reusabilidade; para tanto, é preciso planejamento e disciplina para pensar em termos genéricos, não voltados simplesmente para a aplicação corrente.

## **2.26 O Modelo de Objetos (OMT)**

Um modelo de objetos busca capturar a estrutura estática de um sistema mostrando os objetos existentes, seus relacionamentos, atributos e operações que caracterizam cada classe de objetos. É através do uso deste modelo que se enfatiza o desenvolvimento em termos de objetos ao invés de mecanismos tradicionais de desenvolvimento baseado em funcionalidades, permitindo uma representação mais próxima do mundo real.

Uma vez que as principais definições e conceitos da abordagem de orientação a objetos estão definidos, é possível introduzir o modelo de objetos que será adotado ao longo deste texto. O modelo apresentado é um subconjunto do modelo OMT (Object Modeling Technique), proposto por Rumbaugh entre outros. Este modelo também introduz uma representação diagramática para este modelo, a qual será também apresentada aqui.

### **2.26.1 Objetos e Classes**

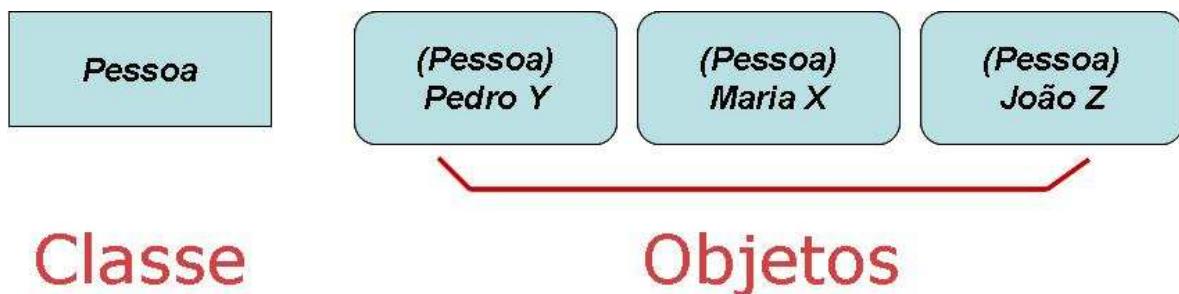
Objeto é definido neste modelo como um conceito, abstração ou coisa com limites e significados bem definidos para a aplicação em questão. Objetos têm dois propósitos: promover o entendimento do mundo real e suportar uma base prática para uma implementação computacional. Não existe uma maneira “correta” de decompor um problema em objetos; esta decomposição depende do julgamento do projetista e da natureza do problema. Todos os objetos têm identidade própria e são distinguíveis.

Uma classe de objetos descreve um grupo de objetos com propriedades (atributos) similares, comportamentos (operações) similares, relacionamentos comuns com outros objetos e uma semântica comum. Por exemplo, Pessoa e Companhia são classes de objetos. Cada pessoa tem um nome e uma idade; estes seriam os atributos comuns da classe. Companhias também podem ter os mesmos atributos nome e idade definidos. Entretanto, devido à distinção semântica, elas provavelmente estariam agrupados em outra classe que não Pessoa. Como se pode observar, o agrupamento em classes não leva em conta apenas o compartilhamento de propriedades.

Todo objeto sabe a que classe ele pertence, ou seja, a classe de um objeto é um atributo implícito do objeto. Este conceito é suportado na maior parte das linguagens de programação orientada a objetos, inclusive em ADVPL.

OMT define dois tipos de diagramas de objetos, diagramas de classes e diagramas de instâncias. Um diagrama de classe é um esquema, ou seja, um padrão ou gabarito que descreve as muitas possíveis instâncias de dados. Um diagrama de instâncias descreve como um conjunto particular de objetos está relacionado.

Diagramas de instâncias são úteis para apresentar exemplos e documentar casos de testes; diagramas de classes têm uso mais amplo. A Figura abaixo apresenta a notação adotada para estes diagramas.



**Figura:** Representação diagramática de OMT para classes e objetos

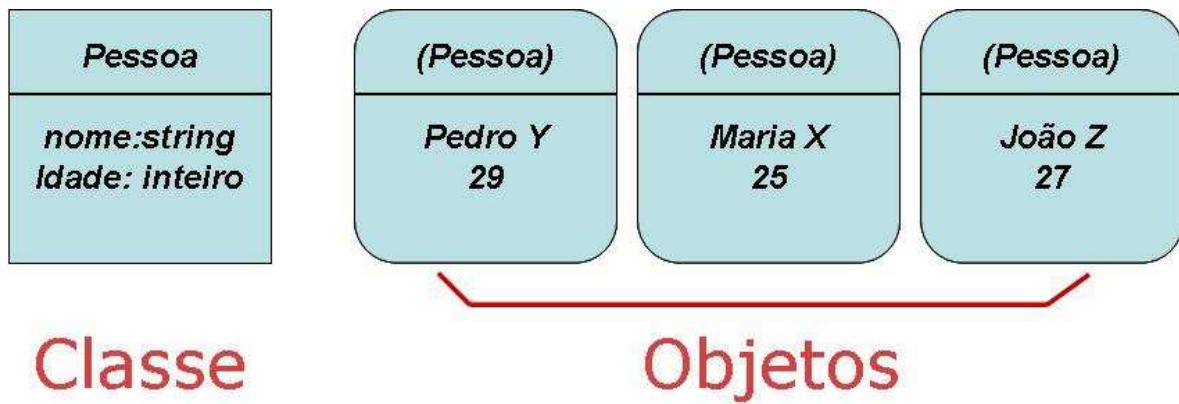
O agrupamento de objetos em classes é um poderoso mecanismo de abstração. Desta forma, é possível generalizar definições comuns para uma classe de objetos, ao invés de repetí-las para cada objeto em particular. Esta é uma das formas de reutilização e economia que a abordagem de orientação a objetos suporta.

### 2.26.2 Atributos

Um atributo é um valor de dado assumido pelos objetos de uma classe. Nome, idade e peso são exemplos de atributos de objetos Pessoa. Cor, peso e modelo são possíveis atributos de objetos Carro. Cada atributo tem um valor para cada instância de objeto. Por exemplo, o atributo idade tem valor ``29'' no objeto Pedro Y. Em outras palavras, Pedro Y tem 29 anos de idade. Diferentes instâncias de objetos podem ter o mesmo valor para um dado atributo.

Cada nome de atributo é único para uma dada classe, mas não necessariamente único entre todas as classes. Por exemplo, ambos Pessoa e Companhia podem ter um atributo chamado endereço.

No diagrama de classes, atributos são listados no segundo segmento da caixa que representa a classe. O nome do atributo pode ser seguido por detalhes opcionais, tais como o tipo de dado assumido e valor default. A Figura abaixo mostra esta representação.



**Figura:** Representação diagramática de OMT para classes e objetos com atributos

Não se deve confundir identificadores internos de objetos com atributos do mundo real. Identificadores de objetos são uma conveniência de implementação, e não têm nenhum significado para o domínio da aplicação. Por exemplo, CIC e RG não são identificadores de objetos, mas sim verdadeiros atributos do mundo real.

### 2.26.3 Operações e Métodos

Uma operação é uma função ou transformação que pode ser aplicada a ou por objetos em uma classe. Por exemplo, abrir, salvar e imprimir são operações que podem ser aplicadas a objetos da classe Arquivo. Todos os objetos em uma classe compartilham as mesmas operações.

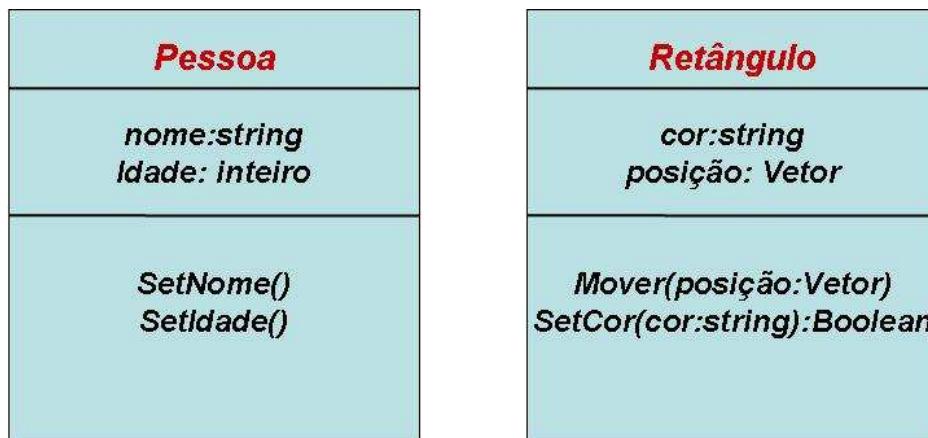
Toda operação tem um objeto-alvo como um argumento implícito. O comportamento de uma operação depende da classe de seu alvo. Como um objeto “sabe” qual sua classe, é possível escolher a implementação correta da operação. Além disto, outros argumentos (parâmetros) podem ser necessários para uma operação.

Uma mesma operação pode se aplicar a diversas classes diferentes. Uma operação como esta é dita ser polimórfica, ou seja, ela pode assumir distintas formas em classes diferentes.

Um método é a implementação de uma operação para uma classe. Por exemplo, a operação imprimir pode ser implementada de forma distinta, dependendo se o arquivo a ser impresso contém apenas texto ASCII, é um arquivo de um processador de texto ou binário. Todos estes métodos executam a mesma operação: imprimir o arquivo; porém, cada método será implementado por um diferente código.

A assinatura de um método é dada pelo número e tipos de argumentos do método, assim como por seu valor de retorno. Uma estratégia de desenvolvimento recomendável é manter assinaturas coerentes para métodos implementando uma dada operação, assim como um comportamento consistente entre as implementações.

Em termos de diagramas OMT, operações são listadas na terceira parte da caixa de uma classe. Cada nome de operação pode ser seguida por detalhes opcionais, tais como lista de argumentos e tipo de retorno. A lista de argumentos é apresentada entre parênteses após o nome da operação. Uma lista de argumentos vazia indica que a operação não tem argumentos; da ausência da lista de argumentos não se pode concluir nada. O tipo de resultado vem após a lista de argumentos, sendo precedido por dois pontos (:). Caso a operação retorne resultado, este não deve ser omitido, pois esta é a forma de distingui-la de operações que não retornam resultado. Exemplos de representação de operações em OMT são apresentados na Figura abaixo:



**Figura:** Representação diagramática de OMT para classes com atributos e operações

---

#### 2.26.4 Sugestões de desenvolvimento

Na construção de um modelo para uma aplicação, as seguintes sugestões devem ser observadas a fim de se obter resultados claros e consistentes:

- Não comece a construir um modelo de objetos simplesmente definindo classes, associações e heranças. A primeira coisa a se fazer é entender o problema a ser resolvido.

- Tente manter seu modelo simples. Evite complicações desnecessárias.

- Escolha nomes cuidadosamente. Nomes são importantes e carregam conotações poderosas. Nomes devem ser descritivos, claros e não deixar ambigüidades. A escolha de bons nomes é um dos aspectos mais difíceis da modelagem.

- Não “enterre” apontadores ou outras referências a objetos dentro de objetos como atributos. Ao invés disto, modele estas referências como associações. Isto torna o modelo mais claro e independente da implementação.

- Tente evitar associações que envolvam três ou mais classes de objetos. Muitas vezes, estes tipos de associações podem ser decompostos em termos de associações binárias, tornando o modelo mais claro.

- Não transfira os atributos de ligação para dentro de uma das classes.

- Tente evitar hierarquias de generalização muito profundas.

- Não se surpreenda se o seu modelo necessitar várias revisões; isto é o normal.
- Sempre documente seus modelos de objetos. O diagrama pode especificar a estrutura do modelo, mas nem sempre é suficiente para descrever as razões por trás da definição do modelo. Uma explicação escrita pode clarificar pontos tais como significado de nomes e explicar a razão para cada classe e relacionamento.
- Nem sempre todas as construções OMT são necessárias para descrever uma aplicação. Use apenas aquelas que forem adequadas para o problema analisado.

## **13. Orientação a objetos em ADVPL**

---

Neste tópico será detalhada a forma com a qual a linguagem ADVPL implementa os conceitos de orientação a objetos e a sintaxe utilizada no desenvolvimento de aplicações.

### **2.27 Sintaxe e operadores para orientação a objetos**

---

#### **Palavras reservadas**

---

 **CLASS**

 **CONSTRUCTOR**

 **DATA**

 **ENDCLASS**

 **FROM**

 **METHOD**

 **SELF**

#### **CLASS**

---

<b>Descrição</b>	Utilizada na declaração de uma classe de objetos, e para identificar a qual classe um determinado método está relacionado.
<b>Sintaxe 1</b>	CLASS <nome_da_classe>
<b>Sintaxe 2</b>	METHOD <nome_do_método> CLASS <nome_da_classe>

#### **CONSTRUCTOR**

---

<b>Descrição</b>	Utilizada na especificação de um método especial, definido como construtor, o qual tem a função de retornar um novo objeto com os atributos e métodos definidos na classe.
------------------	--

<b>Sintaxe</b>	METHOD <nome_do_método()> CONSTRUCTOR
----------------	---------------------------------------

## DATA

---

<b>Descrição</b>	Utilizada na declaração de um atributo da classe de objetos.
<b>Sintaxe</b>	DATA <nome_do_atributo>

## **ENDCLASS**

---

<b>Descrição</b>	Utilizada na finalização da declaração da classe.
<b>Sintaxe</b>	ENDCLASS

## **FROM**

---

<b>Descrição</b>	Utilizada na declaração de uma classe, a qual será uma instância de uma superclasse, recebendo os atributos e métodos nela definidos, implementando a herança entre classes.
<b>Sintaxe</b>	CLASS <nome_da_classe> FROM <nome_da_superclasse>

## **METHOD**

---

<b>Descrição</b>	Utilizada na declaração do protótipo do método de uma classe de objetos, e na declaração do método efetivamente desenvolvido.
<b>Sintaxe 1</b>	METHOD <nome_do_método()>
<b>Sintaxe 2</b>	METHOD <nome_do_método(<parâmetros>)> CLASS <nome_da_classe>

## **SELF**

---

<b>Descrição</b>	Utilizada principalmente pelo método construtor para retornar o objeto criado para a aplicação.
<b>Sintaxe</b>	Return SELF

## **Operadores específicos**

---

:	Utilizado para referenciar um método ou um atributo de um objeto já instanciado.
<b>Exemplo 1</b>	cNome := oAluno:sNome
<b>Exemplo 2</b>	cNota := oAluno:GetNota(cCurso)

::	Utilizado pelos métodos de uma classe para referenciar os atributos disponíveis para o objeto.  METHOD GetNota(cCurso) CLASS ALUNO  Local nPosCurso := 0 Local nNota := 0  Exemplo nPosCurso := aScan(::aCursos,{  aCurso  aCurso[1] == cCurso})  IF nPosCurso > 0
----	--

```
nNota := ::aCursos[nPosCurso][2]
ENDIF
Return nNota
```

## 2.28 Estrutura de uma classe de objetos em ADVPL

### **Declaração da classe**

A declaração de uma classe da linguagem ADVPL é realizada de forma similar a declaração de uma função, com a diferença de que uma classe não possui diferenciação quanto a sua procedência, como uma Function() e uma User Function(), e não possui visibilidade limitada como uma Static Function().

Exemplo:

```
#include "protheus.ch"
CLASS Pessoa()
```

### **Definição dos atributos**

Seguindo o mesmo princípio de variáveis não tipadas, os atributos das classes em ADVPL não precisam ter seu tipo especificado, sendo necessário apenas determinar seus nomes.

Desta forma é recomendado o uso da notação Húngara também para a definição dos atributos de forma a facilitar a análise, interpretação e utilização da classe e seus objetos instanciados.

Exemplo:

```
#include "protheus.ch"
CLASS Pessoa()

DATA cNome
DATA nIdade
```

### **Prototipação dos métodos**

A prototipação dos métodos é uma regra utilizada pelas linguagens orientadas a objetos, através da qual são especificadas as operações que podem ser realizadas pelo objeto, diferenciando os métodos de outras funções internas de uso da classe, e para especificar quais são os métodos construtores.

Em linguagens tipadas, na prototipação dos métodos é necessário definir quais são os parâmetros recebidos e seus respectivos tipos, além de definir o tipo do retorno que será fornecido. Em ADVPL é necessário apenas descrever a chamada do método e caso necessário se o mesmo é um construtor.

Exemplo:

```
#include "protheus.ch"
CLASS Pessoa()

DATA cNome
DATA nldade

METHOD Create() CONSTRUCTOR
METHOD SetNome()
METHOD SetIdade()

ENDCLASS
```

## 2.29 Implementação dos métodos de uma classe em ADVPL

### Método Construtor

O método construtor possui a característica de retornar um objeto com o tipo da classe da qual o mesmo foi instanciado. Por esta razão diz-se que o tipo do objeto instanciado é a classe daquele objeto.

Para produzir este efeito, o método construtor utiliza a palavra reservada “SELF”, a qual é utilizada pela linguagem ADVPL para referência a própria classe daquele objeto.

Exemplo:

```
#include "protheus.ch"
CLASS Pessoa()

DATA cNome
DATA nldade

METHOD Create() CONSTRUCTOR
METHOD SetNome()
METHOD SetIdade()

ENDCLASS

METHOD Create(cNome, nldade) CLASS Pessoa
```

```
::cNome := cNome  
::nIdade := nIdade  
  
Return SELF
```

## **Manipulação de atributos**

---

Os atributos definidos para uma classe com a utilização da palavra reservada “DATA” em sua declaração podem ser manipulados por seus métodos utilizando o operador “::”.

A utilização deste operador permite ao interpretador ADVPL diferenciar variáveis comuns criadas pelas funções e métodos que utilizam este objeto dos atributos propriamente ditos.

Exemplo:

```
#include "protheus.ch"  
CLASS Pessoa()  
  
DATA cNome  
DATA nIdade  
  
METHOD Create() CONSTRUCTOR  
METHOD SetNome()  
METHOD SetnIdade()  
ENDCLASS  
  
METHOD Create(cNome, nIdade) CLASS Pessoa  
  
::cNome := cNome  
::nIdade := nIdade  
  
Return SELF
```

## **Utilização de funções em uma classe de objetos**

Conforme mencionado anteriormente, a utilização da palavra reservada “METHOD” permite ao interpretador ADVPL diferenciar os métodos que podem ser utilizados através da referência do objeto de funções internas descritas internamente na classe.

Isto permite a utilização de funções tradicionais da linguagem ADVPL, como as Static Functions() as quais serão visíveis apenas a classe, e não poderão ser referenciadas diretamente pelo objeto.

Exemplo – parte 01: Função CadPessoa (usuária da classe Pessoa)

```
#include "protheus.ch"

USER FUNCTION CadPessoa()

Local oPessoa
Local cNome := ""
Local dNascimento:= CTOD("")
Local aDados := {}

aDados := GetDados()
oPessoa := Pessoa():Create(cNome,dNascimento)

Return
```

Exemplo – parte 02: Classe Pessoa

```
#include "protheus.ch"
CLASS Pessoa()

DATA cNome
DATA nIdade
DATA dNascimento

METHOD Create() CONSTRUCTOR
METHOD SetNome()
METHOD SetIdade()

ENDCLASS

METHOD Create(cNome, dNascimento) CLASS Pessoa
::cNome := cNome
::dNascimento := dNascimento
::nIdade := CalIdade(dNascimento)
Return SELF

STATIC FUNCTION CalIdade(dNascimento)
Local nIdade
nIdade := dDataBase - dNascimento
RETURN nIdade
```

## Herança entre classes

Seguindo o princípio da orientação a objetos, a linguagem ADVPL permite que uma classe receba por herança os métodos e atributos definidos em uma outra classe, a qual tornasse a superclasse desta instância.

Para utilizar este recurso deve ser utilizada a palavra reservada “FROM” na declaração da classe, especificando a superclasse que será referenciada.

Em ADVPL o exemplo prático desta situação é a superclasse TSrvObject, a qual é utilizada pela maioria das classes e componentes da interface visual, como demonstrado no módulo 06.

Exemplo – parte 01: Declaração da classe Pessoa

```
#include "protheus.ch"
CLASS Pessoa()

DATA cNome
DATA nIdade
DATA dNascimento

METHOD Create() CONSTRUCTOR
METHOD SetNome()
METHOD SetIdade()

ENDCLASS
```

Exemplo – parte 02: Declaração da classe Aluno

```
#include "protheus.ch"
CLASS Aluno() FROM Pessoa

DATA nID
DATA aCursos

METHOD Create() CONSTRUCTOR
METHOD Inscrever()
METHOD Avaliar()
METHOD GetNota()
METHOD GetStatus()

ENDCLASS

// Os objetos da classe Aluno, possuem todos os métodos e atributos
// da classe Pessoa, além
// dos métodos e atributos declarados na própria classe.
```

## **Construtor para classes com herança**

Quanto é utilizado o recurso de herança entre classes, o construtor da classe instanciada deve receber um tratamento adicional, para que o objeto instanciado seja criado com os atributos e métodos definidos na superclasse.

Nestes casos, logo após a definição do método construtor da classe, deverá ser executado o método construtor da superclasse.

Exemplo – parte 03: Método Construtor da classe Aluno

```
METHOD Create(cNome,dNascimento,nID)
:Create(cNome,dNascimento) // Chamada do método construtor da classe
Pessoa.

::nID := ID

Return SELF
```

## **MÓDULO 06: ADVPL ORIENTADO À OBJETOS I**

Neste módulo serão tratados os componentes e objetos da interface visual da linguagem ADVPL, permitindo o desenvolvimento de aplicações com interfaces gráficas com sintaxe orientada a objetos.

### **14. Componentes da interface visual do ADVPL**

A linguagem ADVPL possui diversos componentes visuais e auxiliares, os quais podem ser representados utilizando a estrutura abaixo:

#### **Classes da Interface**

Visual

Tsrvoobject

#### **Classes Auxiliares**

Tfont

#### **Classes de Janelas**

Msdialog

Tdialog

Twindow

#### **Classes de Componentes**

Tcontrol

#### **Classes de Componentes Visuais**

Brgetddb

Mscalend

Mscalendgrid

Msselbr

Msworktime

Sbutton

Tbar

Tbitmap

Tbrowsebutton

Tbtnbmp

Tbtnbmp2

Tbutton

Tcbrowse

Tcheckbox

Tcolortriangle

Tcombobox  
Tfolder  
Tfont  
Tget  
Tgroup  
Thbutton  
Tibrowser  
Tlistbox  
Tmenu  
Tmenubar  
Tmeter  
Tmsgraphic  
Tmsgbar  
Tmultibtn  
Tmultiget  
Tolecontainer  
Tpgeview  
Tpanel  
Tradmenu  
Tsbrowse  
Tsay  
Tscrollbox  
Tsimpleeditor  
Tslider

**Classes de  
Componentes  
Visuais**

Tsplitter  
Ttabs  
Ttoolbox  
Twbrowse  
Vcbrowse

## Classes da interface visual

### **TSRVOBJECT()**

#### **Descrição**

Classe abstrata inicial de todas as classes de interface do ADVPL. Não deve ser instanciada diretamente.

### **Classes auxiliares**

### **TFONT()**

#### **Descrição**

Classe de objetos que define a fonte **do texto utilizado nos controles visuais**.

### **Classes de janelas**

### **MSDIALOG()**

#### **Descrição**

Classe de objetos que deve ser utilizada como padrão de janela para entrada de dados. MSDialog é um tipo de janela diálogo modal, isto é, não permite que outra janela ativa receba dados enquanto esta estiver ativa.

### **TDIALOG()**

#### **Descrição**

Classe de objetos do tipo diálogo de entrada de dados, sendo seu uso reservado. Recomenda-se utilizar a classe MSDialog que é herdada desta classe.

### **TWINDOW()**

#### **Descrição**

Classe de objetos do tipo diálogo principal de programa. Deverá existir apenas uma instância deste objeto na execução do programa.

### **Classes de componentes**

### **TCONTROL()**

#### **Descrição**

Classe abstrata comum entre todos os componentes visuais editáveis. Não deve ser instanciada diretamente.

### **Classes de componentes visuais**

### **BRGETDB()**

#### **Descrição**

Classe de objetos visuais do tipo Grid.

## **MSCALEND()**

<b>Descrição</b>	Classe de objetos visuais do tipo Calendário.
------------------	---

## **MSCALENDGRID()**

<b>Descrição</b>	Classe de objetos visuais do tipo Grade de Períodos.
------------------	--

## **MSSELBR()**

<b>Descrição</b>	Classe de objetos visuais do tipo controle - Grid
------------------	---

## **MSWORKTIME()**

<b>Descrição</b>	Classe de objetos visuais do tipo controle - Barra de Período.
------------------	--

## **SBUTTON()**

<b>Descrição</b>	Classe de objetos visuais do tipo botão, o qual pode possuir imagens padrões associadas ao seu tipo.
------------------	--

## **TBAR()**

<b>Descrição</b>	Classe de objetos visuais do tipo Barra Superior.
------------------	---

## **TBITMAP()**

<b>Descrição</b>	Classe de objetos visuais que permite a exibição de uma imagem.
------------------	---

## **TBROWSEBUTTON()**

<b>Descrição</b>	Classe de objetos visuais do tipo botão no formato padrão utilizado em browses da aplicação.
------------------	--

## **TBTNBMP()**

<b>Descrição</b>	Classe de objetos visuais do tipo botão, o qual permite que seja vinculada uma imagem ao controle.
------------------	--

## **TBTNBMP2()**

<b>Descrição</b>	Classe de objetos visuais do tipo botão, o qual permite a exibição de uma imagem ou de um popup.
------------------	--

**TBUTTON()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo botão, o qual permite a utilização de texto para sua identificação.
------------------	---

**TCBROWSE()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo Grid.
------------------	---

**TCHECKBOX()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle - CheckBox.
------------------	--

**TCOLORTRIANGLE()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo Paleta de Cores.
------------------	--

**TCOMBOBOX()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo tComboBox, a qual cria uma entrada de dados com múltipla escolha com item definido em uma lista vertical, acionada por F4 ou pelo botão esquerdo localizado na parte direita do controle. A variável associada ao controle terá o valor de um dos itens selecionados ou no caso de uma lista indexada, o valor de seu índice.
------------------	---

**TFOLDER()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle - Folder.
------------------	--

**TGET()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle – tGet, a qual cria um controle que armazena ou altera o conteúdo de uma variável através de digitação. O conteúdo da variável só é modificado quando o controle perde o foco de edição para outro controle.
------------------	---

**TGROUP()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo painel – tGroup, a qual cria um painel onde controles visuais podem ser agrupados ou classificados. Neste painel é criada uma borda com título em volta dos controles agrupados.
------------------	--

**THBUTTON()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo botão com hiperlink.
------------------	--

**TIBROWSER()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo Página de Internet, sendo necessário incluir a clausula BrowserEnabled=1 no Config do Remote.INI
------------------	--

**TLISTBOX()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle – tListbox, a qual cria uma janela com itens selecionáveis e barra de rolagem. Ao selecionar um item, uma variável é atualizada com o conteúdo do item selecionado.
------------------	--

**TMENU()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle - Menu.
------------------	--

**TMENUBAR()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle - Barra de Menu.
------------------	---

**TMETER()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle – tMeter, a qual exibe uma régua (gauge) de processamento, descrevendo o andamento de um processo através da exibição de uma barra horizontal.
------------------	---

**TMSGGRAPHIC()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle - Gráfico.
------------------	---

**TMSGBAR()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle - Rodapé.
------------------	--

**TMULTIBTN()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle - Múltiplos botões.
------------------	--

**TMULTIGET()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle - edição de texto de múltiplas linhas.
------------------	---

**TOLECONTAINER()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle, a qual permite a criação de um botão vinculado a um objeto OLE.
------------------	---

**TPAGEVIEW()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle, que permite a visualização de arquivos no formato gerado pelo spool de impressão do Protheus.
------------------	---

**TPANEL()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle – tPanel, a qual permite criar um painel estático, onde podem ser criados outros controles com o objetivo de organizar ou agrupar componentes visuais.
------------------	---

**TRADMENU()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle – TRadMenu, a qual permite criar um controle visual no formato Radio Button.
------------------	---

**TSBROWSE()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle – TSBrowse, a qual permite criar um controle visual do tipo Grid.
------------------	--

**TSAY()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle – tSay, a qual exibe o conteúdo de texto estático sobre uma janela ou controle previamente definidos.
------------------	--

**TSCROLLBOX()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle – tScrollbox, a qual permite criar um painel com scroll deslizantes nas laterais (horizontais e verticais) do controle.
------------------	--

## **TSIMPLEEDITOR()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle – tSimpleEditor, a qual permite criar um controle visual para edição de textos com recursos simples, como o NotePad®
------------------	---

## **TSLIDER()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle – tSlider, a qual permite criar um controle visual do tipo botão deslizante.
------------------	---

## **TSPLITTER()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle – tSplitter, a qual permite criar um controle visual do tipo divisor.
------------------	--

## **TTABS()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle – TTabs, a qual permite criar um controle visual do tipo pasta.
------------------	--

## **TTOOLBOX()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle – tToolbox, a qual permite criar um controle visual para agrupar diferentes objetos.
------------------	---

## **TWBROWSE()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle – TWBrowse, a qual permite criar um controle visual do tipo Grid.
------------------	--

## **VCBROWSE()**

---

<b>Descrição</b>	Classe de objetos visuais do tipo controle – VCBrowse, a qual permite criar um controle visual do tipo Grid.
------------------	--

## **Documentação dos componentes da interface visual**

---

Os componentes da interface visual da linguagem ADVPL utilizados neste treinamento estão documentados na seção Guia de Referência, ao final deste material.

## 2.30 Particularidades dos componentes visuais

### 2.30.1 Configurando as cores para os componentes

Os componentes visuais da linguagem ADVPL utilizam o padrão de cores RGB.

As cores deste padrão são definidas pela seguinte fórmula, a qual deve ser avaliada tendo como base a paleta de cores no formato RGB:

$$nCor := nVermelho + (nVerde * 256) + (nAzul * 65536)$$

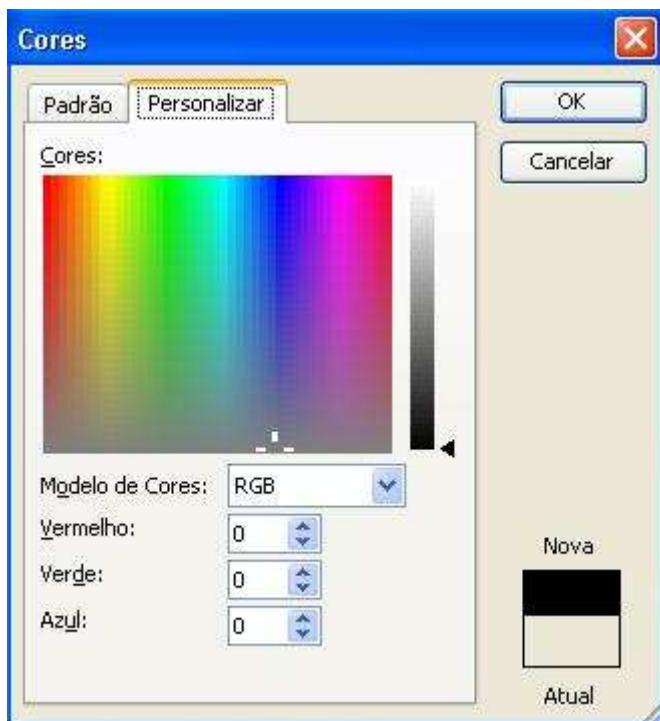


Figura: Paleta de cores no formato RGB

Com base nesta paleta, podemos definir os valores das seguintes cores básicas:

Cor	R	G	B	Valor
Preto	0	0	0	0
Azul	0	0	255	16711680
Verde	0	255	0	65280
Ciano	0	255	255	16776960
Vermelho	255	0	0	255
Rosa	255	0	255	16711935
Amarelo	255	255	0	65535
Branco	255	255	255	16777215

Para atribuir as cores aos objetos visuais devem ser observados os atributos utilizados para estes fins em cada objeto, como por exemplo:

#### **MSDIALOG()**

---

<b>nClrPane</b>	Cor de fundo do painel
<b>nClrText</b>	Cor da fonte das letras do painel

#### **TSAY()**

---

<b>nClrPane</b>	Cor de fundo do painel
<b>nClrText</b>	Cor da fonte das letras do painel

#### **Função RGB()**

---

A linguagem ADVPL possui a função RGB() a qual retorna o valor da cor a ser definido, de acordo com a parametrização de cada um dos elementos da paleta RGB.

#### **RGB(nRed, nGreen, nBlue)**

---

<b>nRed</b>	Valor de 0-255 para o elemento vermelho da paleta RGB
<b>nGreen</b>	Valor de 0-255 para o elemento verde da paleta RGB
<b>nBlue</b>	Valor de 0-255 para o elemento azul da paleta RGB
<b>Retorno</b>	Valor a ser definido para o atributo cor do componente

### **15. Aplicações com a interface visual do ADVPL**

---

A linguagem ADVPL possui interfaces visuais pré-definidas que auxiliam no desenvolvimento de aplicações mais completas, combinando estas interfaces com os componentes visuais demonstrados anteriormente.

Didaticamente as interfaces visuais pré-definidas da linguagem ADVPL podem ser divididas em três grupos:

- **Captura de informações simples ou Multi-Gets;**
- **Captura de múltiplas informações ou Multi-Lines;**
- **Barras de botões**

## 2.31 Captura de informações simples (Multi-Gets)

Em ADVPL, as telas de captura de informações compostas por múltiplos campos digitáveis acompanhados de seus respectivos textos explicativos são comumente chamados de Enchoices.

Um Enchoice pode ser facilmente entendida como diversos conjuntos de objetos TSay e TGet alinhados de forma a visualizar ou capturar informações, normalmente vinculadas a arquivos de cadastros ou movimentações simples.

Abaixo temos a visualização de uma Enchoice para o arquivo padrão do ERP Protheus de Cadastro de Clientes ("SA1"):

The screenshot shows a Windows application window titled "Cadastrais" under the "Clientes - Incluir" tab. The window contains several groups of input fields with associated descriptive text and validation logic. For example, the "Nome" field has the text "Nome" above it, and the "Endereço" field has "Endereço" above it. There are also dropdown menus and search icons (magnifying glass) for some fields like "Tipo" and "País".

**Figura: Enchoice do Cadastro de Clientes do ERP Protheus**

A linguagem ADVPL permite a implementação da Enchoice de duas formas similares:

- Função Enchoice:** Sintaxe tradicionalmente utilizada em ADVPL, a qual não retorna um objeto para a aplicação chamadora;
- Classe MsMGet:** Classe do objeto Enchoice, a qual permite a instanciação direta de um objeto, tornando-o disponível na aplicação chamadora.

A utilização de um ou outro objeto depende unicamente da escolha do desenvolvedor já que os parâmetros para a função Enchoice e para o método New() da classe MsMGet

são os mesmos, lembrando que para manter a coerência com uma aplicação escrita em orientação a objetos deverá ser utilizada a classe MsMGet().

### 2.31.1 Enchoice()

**Sintaxe:** Enchoice( **cAlias**, **nReg**, **nOpc**, **aCRA**, **cLetra**, **cTexto**, **aAcho**, **aPos**,  
**aCpos**, **nModelo**, **nColMens**, **cMensagem**, **cTudoOk**, **oWnd**, **IF3**,  
**IMemoria**, **IColumn**, **caTela**, **INoFolder**, **IProperty**)

**Retorno:** Nil

**Parâmetros:**

<b>cAlias</b>	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada
<b>nReg</b>	Parâmetro não utilizado
<b>nOpc</b>	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização)
<b>aCRA</b>	Parâmetro não utilizado
<b>cLetra</b>	Parâmetro não utilizado
<b>cTexto</b>	Parâmetro não utilizado
<b>aAcho</b>	Vetor com nome dos campos que serão exibidos. Os campos de usuário sempre serão exibidos se não existir no parâmetro um elemento com a expressão "NOUSER"
<b>aPos</b>	Vetor com coordenadas para criação da enchoice no formato {<top>, <left>, <bottom>, <right>}
<b>aCpos</b>	Vetor com nome dos campos que poderão ser editados
<b>nModelo</b>	Se for diferente de 1 desabilita execução de gatilhos estrangeiros
<b>nColMens</b>	Parâmetro não utilizado
<b>cMensagem</b>	Parâmetro não utilizado
<b>cTudoOk</b>	Expressão para validação da Enchoice
<b>oWnd</b>	Objeto (janela, painel, etc.) onde a enchoice será criada.
<b>IF3</b>	Indica se a enchoice esta sendo criada em uma consulta F3 para utilizar variáveis de memória
<b>IMemoria</b>	Indica se a enchoice utilizará variáveis de memória ou os campos da tabela na edição
<b>IColumn</b>	Indica se a apresentação dos campos será em forma de coluna
<b>caTela</b>	Nome da variável tipo "private" que a enchoice utilizará no lugar da propriedade aTela
<b>INoFolder</b>	Indica se a enchoice não irá utilizar as Pastas de Cadastro (SXA)
<b>IProperty</b>	Indica se a enchoice não utilizará as variáveis aTela e aGets, somente suas propriedades com os mesmos nomes

## Exemplo: Utilização da função Enchoice()

---

```
#include "protheus.ch"

/*
+-----
| Função      | MBRWENCH   | Autor | ARNALDO RAYMUNDO JR.| Data |
|             |             |       |                     |
+-----+
| Descrição      | Programa que demonstra a utilização da função
Enchoice()|
+-----+
| Uso          | Curso ADVPL
|             |             |
+-----+
/*

User Function MrbwEnch()

Private cCadastro           := " Cadastro de Clientes"
Private aRotina := [{"Pesquisar" , "axPesqui" , 0, 1}; {"Visualizar" , "U_ModEnc" , 0, 2}]

DbSelectArea("SA1")
DbSetOrder(1)

MBrowse(6,1,22,75,"SA1")

Return

User Function ModEnc(cAlias,nReg,nOpc)

Local aCpoEnch      := 0
Local aAlter        := 0

Local cAliasE     := cAlias
Local aAlterEnch := {}
Local aPos         := {000,000,400,600}
Local nModelo      := 3
Local IF3          := .F.
Local IMemoria    := .T.
Local IColumn      := .F.
Local caTela       := ""
Local INoFolder := .F.
Local IProperty := .F.
```

```

Private oDlg
Private oGetD
Private oEnch
Private aTELA[0][0]
Private aGETS[0]

DbSelectArea("SX3")
DbSetOrder(1)
DbSeek(cAliasE)

While !Eof() .And. SX3->X3_ARQUIVO == cAliasE
    If !(SX3->X3_CAMPO $ "A1_FILIAL") .And. cNivel >= SX3->X3_NIVEL
    .And.;

        X3Uso(SX3->X3_USADO)
        AADD(aCpoEnch,SX3->X3_CAMPO)
        EndIf
        DbSkip()
    End

aAlterEnch := aClone(aCpoEnch)

DEFINE MSDIALOG oDlg TITLE cCadastro FROM 000,000 TO 400,600 PIXEL
    RegToMemory("SA1", If(nOpc==3,.T.,.F.))

    Enchoice(cAliasE, nReg, nOpc, /*aCRA*/, /*cLetra*/, /*cTexto*/,
    ;
        aCpoEnch, aPos, aAlterEnch, nModelo, /*nColMens*/;;
        /*cMensagem*/, /*cTodoOk*/, oDlg, IF3, IMemoria,
    lColumn;;
        caTela, INoFolder, IProperty)

ACTIVATE MSDIALOG oDlg CENTERED

Return

```

### 2.31.2 MsMGet()

**Sintaxe:** **MsMGet():New( cAlias, nReg, nOpc, aCRA, cLetra, cTexto, aAcho,**  
**aPos, aCpos, nModelo, nColMens, cMensagem, cTudoOk, oWnd,**  
**IF3, IMemoria, lColumn, caTela, INoFolder, IProperty)**

**Retorno:** oMsMGet é objeto do tipo MsMGet()

**Parâmetros:**

<b>cAlias</b>	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada
---------------	--

<b>nReg</b>	Parâmetro não utilizado
<b>nOpc</b>	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização)
<b>aCRA</b>	Parâmetro não utilizado
<b>cLetra</b>	Parâmetro não utilizado
<b>cTexto</b>	Parâmetro não utilizado
<b>aAcho</b>	Vetor com nome dos campos que serão exibidos. Os campos de usuário sempre serão exibidos se não existir no parâmetro um elemento com a expressão "NOUSER"
<b>aPos</b>	Vetor com coordenadas para criação da enchoice no formato {<top>, <left>, <bottom>, <right>}
<b>aCpos</b>	Vetor com nome dos campos que poderão ser editados
<b>nModelo</b>	Se for diferente de 1 desabilita execução de gatilhos estrangeiros
<b>nColMens</b>	Parâmetro não utilizado
<b>cMensagem</b>	Parâmetro não utilizado
<b>cTudoOk</b>	Expressão para validação da Enchoice
<b>oWnd</b>	Objeto (janela, painel, etc.) onde a enchoice será criada.
<b>IF3</b>	Indica se a enchoice esta sendo criada em uma consulta F3 para utilizar variáveis de memória
<b>IMemoria</b>	Indica se a enchoice utilizará variáveis de memória ou os campos da tabela na edição
<b>IColumn</b>	Indica se a apresentação dos campos será em forma de coluna
<b>caTela</b>	Nome da variável tipo "private" que a enchoice utilizará no lugar da propriedade aTela
<b>INoFolder</b>	Indica se a enchoice não irá utilizar as Pastas de Cadastro (SXA)
<b>IProperty</b>	Indica se a enchoice não utilizará as variáveis aTela e aGets, somente suas propriedades com os mesmos nomes

#### Exemplo: Utilização do objeto MsMGet()

---

```
#include "protheus.ch"

/*
+-----
| Função      | MBRWMSGET | Autor | ARNALDO RAYMUNDO JR. | Data |
|             |           |       |                   |       |
+-----+
| Descrição    | Programa que demonstra a utilização do objeto
MsMget()|
+-----+
```

```

| Uso          | Curso ADVPL
|
+-----+
-----
/*

User Function MrbwMsGet()

Private cCadastro      := " Cadastro de Clientes"
Private aRotina :=      {"Pesquisar" , "axPesqui" , 0, 1};;
                           {"Visualizar" , "U_ModEnc" , 0, 2};

DbSelectArea("SA1")
DbSetOrder(1)

MBrowse(6,1,22,75,"SA1")

Return

User Function ModEnc(cAlias,nReg,nOpc)
Local aCpoEnch      := {}
Local aAlter        := {}

Local cAliasE       := cAlias
Local aAlterEnch := {}
Local aPos   := {000,000,400,600}
Local nModelo      := 3
Local lF3     := .F.
Local lMemoria    := .T.
Local lColumn      := .F.
Local caTela        := ""
Local lNoFolder := .F.
Local lProperty := .F.
Private oDlg
Private oGetD
Private oEnch
Private aTEL[0][0]
Private aGETS[0]

DbSelectArea("SX3")
DbSetOrder(1)
DbSeek(cAliasE)

While !Eof() .And. SX3->X3_ARQUIVO == cAliasE
  If !(SX3->X3_CAMPO $ "A1_FILIAL") .And. cNivel >= SX3->X3_NIVEL
  .And. X3Uso(SX3->X3_USADO)
    AADD(aCpoEnch,SX3->X3_CAMPO)
    EndIf
  DbSkip()

```

```

End

aAlterEnch := aClone(aCpoEnch)

oDlg      := MSDIALOG():New(000,000,400,600,cCadastro,,,,,,.T.)

RegToMemory(cAliasE, If(nOpc==3,.T.,.F.))

oEnch := MsMGet():New(cAliasE, nReg, nOpc, /*aCRA*/,
/*cLetra*/;;
/*cTexto*/, aCpoEnch, aPos, aAlterEnch, nModelo,
/*nColMens*/;;
/*cMensagem*/, /*cTudoOk*/, oDlg, IF3,IMemoria,IColumn,
caTela,;
INoFolder, IProperty)

oDlg:ICentered := .T.
oDlg:Activate()

Return

```

## **2.32 Captura de múltiplas informações (Multi-Lines)**

A linguagem ADVPL permite a utilização de basicamente dois tipos de objetos do tipo grid, ou como também são conhecidos: multi-line:

- **Grids digitáveis:** permitem a visualização e captura de informações, comumente utilizados em interfaces de cadastro e manutenção, tais como:

- **MSGETDB()**
- **MSGETDADOS()**
- **MSNEWGETDADOS()**

- **Grids não digitáveis:** permitem somente a visualização de informações, comumente utilizados como browses do ERP Protheus, tais como:

- **TWBROWSE()**
- **MAWNDBROWSE()**
- **MBROWSE()**

Neste tópico serão tratadas as grids digitáveis disponíveis na linguagem ADVPL para o desenvolvimento de interfaces de cadastros e manutenção de informações.

## **2.32.1 MsGetDB()**

A classe de objetos visuais MsGetDB() permite a criação de um grid digitável com uma ou mais colunas, baseado em uma tabela temporária.

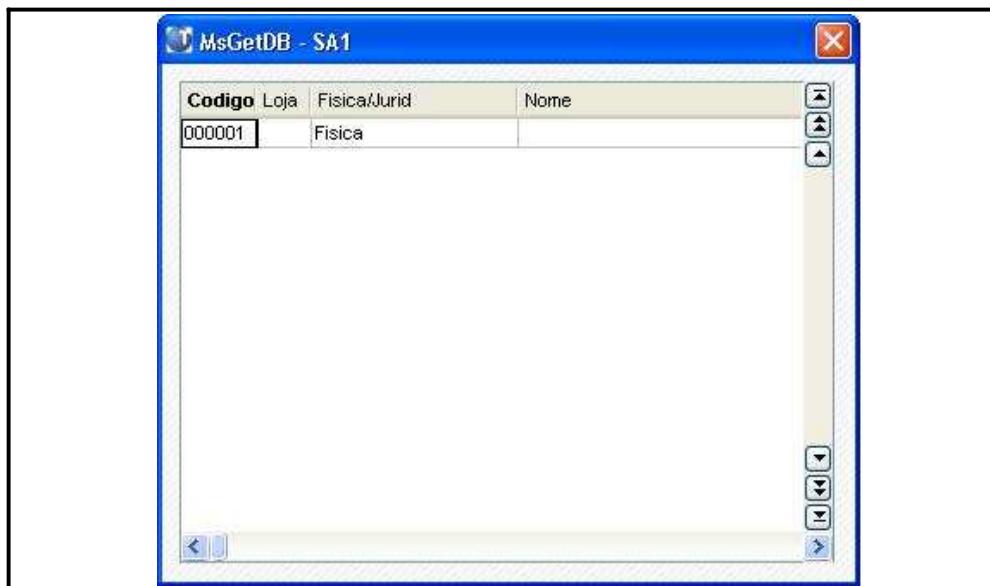
**Sintaxe:** `MsGetDB():New(nTop, nLeft, nBottom, nRight, nOpc, cLinhaOk, cTudoOk, cIniCpos, lDelete, aAlter, nFreeze, lEmpty, uPar1, cTRB, cFieldOk, lCondicional, lAppend, oWnd, lDisparos, uPar2, cDelOk, cSuperDel)`

**Retorno:** `oMsGetDB` objeto do tipo MsGetDB()

**Parâmetros:**

<b>nTop</b>	Distancia entre a MsGetDB e o extremidade superior do objeto que a contém.
<b>nLeft</b>	Distancia entre a MsGetDB e o extremidade esquerda do objeto que a contém.
<b>nBottom</b>	Distancia entre a MsGetDB e o extremidade inferior do objeto que a contém.
<b>nRight</b>	Distancia entre a MsGetDB e o extremidade direita do objeto que a contém.
<b>nOpc</b>	Posição do elemento do vetor aRotina que a MsGetDB usará como referência.
<b>cLinhaOk</b>	Função executada para validar o contexto da linha atual do aCols.
<b>cTudoOk</b>	Função executada para validar o contexto geral da MsGetDB (todo aCols).
<b>cIniCpos</b>	Nome dos campos do tipo caracter que utilizarão incremento automático. Este parâmetro deve ser no formato “+<nome do primeiro campo>+<nome do segundo campo>+...”.
<b>IDelete</b>	Habilita a opção de excluir linhas do aCols. Valor padrão falso.
<b>aAlter</b>	Vetor com os campos que poderão ser alterados.
<b>nFreeze</b>	Indica qual coluna não ficara congelada na exibição.
<b>IsEmpty</b>	Habilita validação da primeira coluna do aCols para esta não poder estar vazia. Valor padrão falso.
<b>uPar1</b>	Parâmetro reservado.
<b>cFieldOk</b>	Função executada na validação do campo.
<b>ctrB</b>	Alias da tabela temporária.
<b>ICondicional</b>	Reservado
<b>IAppend</b>	Indica se a MsGetDB irá criar uma linha em branco automaticamente quando for inclusão.
<b>cDelOk</b>	Função executada para validar a exclusão de uma linha do aCols.
<b>IDisparos</b>	Indica se será utilizado o Dicionário de Dados para consulta padrão, inicialização padrão e gatilhos.
<b>uPar2</b>	Parâmetro reservado.
<b>cSuperDel</b>	-Função executada quando pressionada as teclas <Ctrl>+<Delete>.
<b>oWnd</b>	Objeto no qual a MsGetDB será criada.

**Aparência:**



**Variáveis private:**

<b>aRotina</b>	Vetor com as rotinas que serão executadas na MBrowse e que definira o tipo de operação que esta sendo executada (inclusão, alteração, exclusão, visualização, pesquisa, ...) no formato:  {cTitulo, cRotina, nOpção, nAcesso}, aonde:  nOpção segue o padrão do ERP Protheus para:  1- Pesquisar 2- Visualizar 3- Incluir 4- Alterar 5- Excluir
<b>aHeader</b>	Vetor com informações das colunas no formato:  {cTitulo, cCampo, cPicture, nTamanho, nDecimais,; cValidação, cReservado, cTipo, xReservado1, xReservado2}  A tabela temporária utilizada pela MsGetDB deverá ser criada com base no aHeader mais um último campo tipo lógico que determina se a linha foi excluída.
<b>lRefresh</b>	Variável tipo lógica para uso reservado.

### Variáveis públicas:

<b>nBrLin</b>	Indica qual a linha posicionada do aCols.
---------------	---

### Funções de validação:

<b>cLinhaOk</b>	Função de validação na mudança das linhas da grid. Não pode ser definida como Static Function.
<b>cTudoOk</b>	Função de validação da confirmação da operação com o grid. Não pode ser definida como Static Function.

### Métodos adicionais:

<b>ForceRefresh()</b>	Atualiza a MsGetDB com a tabela e posiciona na primeira linha.
-----------------------	--

### Exemplo: Utilização do objeto MsGetDB()

---

```
#include "protheus.ch"

/*
+-----
| Função      | GETDBSA1      | Autor | MICROSIGA          | Data |
|             |               |       |                      |
+-----+
| Descrição    | Programa que demonstra a utilização do objeto
MsGetDB()|
+-----+
| Uso         | Curso ADVPL
|             |
+-----+
*/
User Function GetDbSA1()

Local nl
Local oDlg
Local oGetDB
Local nUsado := 0
Local aStruct := {}

Private lRefresh := .T.
Private aHeader := {}
```

```

Private aCols := {}

Private aRotina := {"Pesquisar", "AxPesqui", 0, 1},;
                    {"Visualizar", "AxVisual", 0, 2},;
                    {"Incluir", "AxInclui", 0, 3},;
                    {"Alterar", "AxAltera", 0, 4},;
                    {"Excluir", "AxDelete", 0, 5}

DbSelectArea("SX3")
DbSetOrder(1)
DbSeek("SA1")

Exemplo (continuação):

While !Eof() .and. SX3->X3_ARQUIVO == "SA1"
If X3Uso(SX3->X3_USADO) .and. cNivel >= SX3->X3_NIVEL
    nUsado++
    AADD(aHeader,{Trim(X3Titulo()),;
                  SX3->X3_CAMPO,;
                  SX3->X3_PICTURE,;
                  SX3->X3_TAMANHO,;
                  SX3->X3_DECIMAL,;
                  SX3->X3_VALID,;
                  "" ,;
                  SX3->X3_TIPO,;
                  "" ,;
                  "" })
    AADD(aStruct,{SX3->X3_CAMPO,SX3->X3_TIPO,SX3->X3_TAMANHO,;
                  SX3->X3_DECIMAL})
EndIf
DbSkip()
End

AADD(aStruct,{"FLAG","L",1,0})

cCriaTrab := CriaTrab(aStruct,.T.)
DbUseArea(.T.,__LocalDriver,cCriaTrab,,.T.,.F.)

oDlg      := MSDIALOG():New(000,000,300,400, "MsGetDB -
SA1",,,,,,,.T.)
oGetDB := MsGetDB():New(05,05,145,195,3,"U_LINHAOK", "U_TUDOOK",
"+A1_COD",;
.T.,{"A1_NOME"},1,.F.,,cCriaTrab,"U_FIELDOK",,.T.,oDlg,.T.,
,"U_DELOK",;
"U_SUPERDEL")

oDlg:ICentered := .T.
oDlg:Activate()
DbSelectArea(cCriaTrab)
DbCloseArea()

```

```
Return

User Function LINHAOK()
ApMsgStop("LINHAOK")
Return .T.

User Function TUDOOK()
ApMsgStop("LINHAOK")
Return .T.

User Function DELOK()
ApMsgStop("DELOK")
Return .T.

User Function SUPERDEL()
ApMsgStop("SUPERDEL")
Return .T.

User Function FIELDOK()
ApMsgStop("FIELDOK")
Return .T.
```

## **2.32.2 MsGetDados()**

A classe de objetos visuais MsGetDados() permite a criação de um grid digitável com uma ou mais colunas, baseado em um array.

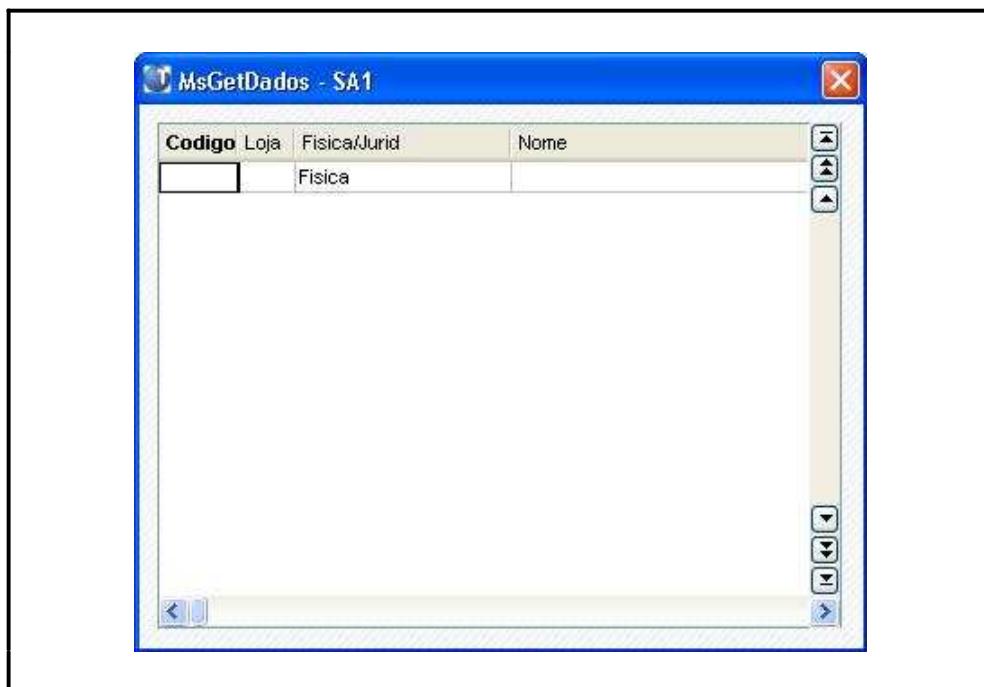
**Sintaxe:** **MsGetDados():New( nTop, nLeft, nBottom, nRight, nOpc,  
cLinhaOk, cTudoOk, cIniCpos, lDelete, aAlter, uPar1, lEmpty,  
nMax, cFieldOk, cSuperDel, uPar2, cDelOk, oWnd)**

**Retorno:** **oMsGetDados** → objeto do tipo **MsGetDados()**

**Parâmetros:**

<b>nTop</b>	Distância entre a MsGetDados e o extremidade superior do objeto que a contém.
<b>nLeft</b>	Distância entre a MsGetDados e o extremidade esquerda do objeto que a contém.
<b>nBottom</b>	Distância entre a MsGetDados e o extremidade inferior do objeto que a contém.
<b>nRight</b>	Distância entre a MsGetDados e o extremidade direita do objeto que a contém.
<b>nOpc</b>	Posição do elemento do vetor aRotina que a MsGetDados usará como referência.
<b>cLinhaOk</b>	Função executada para validar o contexto da linha atual do aCols.
<b>cTudoOk</b>	Função executada para validar o contexto geral da MsGetDados (todo aCols).
<b>cIniCpos</b>	Nome dos campos do tipo caracter que utilizarão incremento automático. Este parâmetro deve ser no formato “+<nome do primeiro campo>+<nome do segundo campo>+...”.
<b>lDelete</b>	Habilita excluir linhas do aCols. Valor padrão falso.
<b>aAlter</b>	Vetor com os campos que poderão ser alterados.
<b>uPar1</b>	Parâmetro reservado.
<b>lEmpty</b>	Habilita validação da primeira coluna do aCols para esta não poder estar vazia. Valor padrão falso.
<b>nMax</b>	Número máximo de linhas permitidas. Valor padrão 99.
<b>cFieldOk</b>	Função executada na validação do campo.
<b>cSuperDel</b>	Função executada quando pressionada as teclas <Ctrl>+<Delete>.
<b>uPar2</b>	Parâmetro reservado.
<b>cDelOk</b>	Função executada para validar a exclusão de uma linha do aCols.
<b>oWnd</b>	Objeto no qual a MsGetDados será criada.

**Aparência:**



**Variáveis private:**

<b>aRotina</b>	Vetor com as rotinas que serão executadas na MBrowse e que definira o tipo de operação que esta sendo executada (inclusão, alteração, exclusão, visualização, pesquisa, ...) no formato: {cTitulo, cRotina, nOpção, nAcesso}, aonde: nOpção segue o padrão do ERP Protheus para:  6- Pesquisar 7- Visualizar 8- Incluir 9- Alterar 10- Excluir
<b>aHeader</b>	Vetor com informações das colunas no formato:  {cTitulo, cCampo, cPicture, nTamanho, nDecimais,; cValidação, cReservado, cTipo, xReservado1, xReservado2}  A tabela temporária utilizada pela MsGetDB deverá ser criada com base no aHeader mais um último campo tipo lógico que determina se a linha foi excluída.

<b>IRefresh</b>	Variável tipo lógica para uso reservado.
-----------------	--

**Variáveis públicas:**

<b>N</b>	Indica qual a linha posicionada do aCols.
----------	---

**Funções de validação:**

<b>cLinhaOk</b>	Função de validação na mudança das linhas da grid. Não pode ser definida como Static Function.
<b>cTudoOk</b>	Função de validação da confirmação da operação com o grid. Não pode ser definida como Static Function.

**Métodos adicionais:**

<b>ForceRefresh()</b>	Atualiza a MsGetDados com a tabela e posiciona na primeira linha.
<b>Hide()</b>	Oculta a MsGetDados.
<b>Show()</b>	Mostra a MsGetDados.

**Exemplo: Utilização do objeto MsGetDados()**

---

```
#include "protheus.ch"

/*
+-----
| Função      | GETDADOSA1 | Autor | MICROSIGA           | Data |
|             |            |        |                      |
+-----+
| Descrição   | Programa que demonstra a utilização do objeto
MSGETADOS()|
+-----+
| Uso         | Curso ADVPL
|             |
+-----+
*/
User Function GetDadoSA1()

Local nl
Local oDlg
Local oGetDados
Local nUsado := 0
Private IRefresh := .T.
```

```

Private aHeader := {}
Private aCols := {}

Private aRotina := {"Pesquisar", "AxPesqui", 0, 1},;
                  {"Visualizar", "AxVisual", 0, 2},;
                  {"Incluir", "AxInclui", 0, 3},;
                  {"Alterar", "AxAltera", 0, 4},;
                  {"Excluir", "AxDelete", 0, 5}

DbSelectArea("SX3")
DbSetOrder(1)
DbSeek("SA1")

While !Eof() .and. SX3->X3_ARQUIVO == "SA1"
If X3Uso(SX3->X3_USADO) .and. cNivel >= SX3->X3_NIVEL
    nUsado++
    AADD(aHeader,{Trim(X3Titulo()),;
                  SX3->X3_CAMPO,;
                  SX3->X3_PICTURE,;
                  SX3->X3_TAMANHO,;
                  SX3->X3_DECIMAL,;
                  SX3->X3_VALID,;
                  "" ,;
                  SX3->X3_TIPO,;
                  "" ,;
                  "" })
EndIf
DbSkip()
End

AADD(aCols,Array(nUsado+1))

For nl := 1 To nUsado
    aCols[1][nl] := CriaVar(aHeader[nl][2])
Next

aCols[1][nUsado+1] := .F.

oDlg      := MSDIALOG():New(000,000,300,400, "MsGetDados -
SA1",,,,,,,.T.)

oGetDados := MsGetDados():New(05, 05, 145, 195, 4, "U_LINHAOK",
"U_TUDOOK",;
"+A1_COD", .T., {"A1_NOME"}, , .F., 200, "U_FIELDOK",
"U_SUPERDEL",;
"U_DELOK", oDlg)

oDlg:lCentered := .T.
oDlg:Activate()

```

```

Return

User Function LINHAOK()
ApMsgStop("LINHAOK")
Return .T.

User Function TUDOOK()
ApMsgStop("LINHAOK")
Return .T.

User Function DELOK()
ApMsgStop("DELOK")
Return .T.

User Function SUPERDEL()
ApMsgStop("SUPERDEL")
Return .T.

User Function FIELDOK()
ApMsgStop("FIELDOK")
Return .T.

```

### 2.32.3 MsNewGetDados()

A classe de objetos visuais MsNewGetDados() permite a criação de um grid digitável com uma ou mais colunas, baseado em um array.

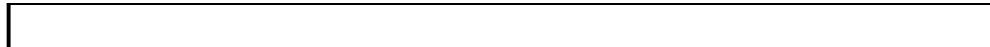
**Sintaxe:** `MsNewGetDados():New(nSuperior, nEsquerda ,nInferior, nDireita, nOpc, cLinOk, cTudoOk, clniCpos, aAlterGDa, nFreeze, nMax, cFieldOk, cSuperDel, cDelOk, oDLG, aHeader, aCols)`

**Retorno:** `oMsGetDados` → objeto do tipo `MsNewGetDados()`

**Parâmetros:**

<b>nSuperior</b>	Distancia entre a MsNewGetDados e o extremidade superior do objeto que a contem
<b>nEsquerda</b>	Distancia entre a MsNewGetDados e o extremidade esquerda do objeto que a contem
<b>nInferior</b>	Distancia entre a MsNewGetDados e o extremidade inferior do objeto que a contem
<b>nDireita</b>	Distancia entre a MsNewGetDados e o extremidade direita do objeto que a contem
<b>nOpc</b>	Operação em execução: 2- Visualizar, 3- Incluir, 4- Alterar, 5- Excluir

<b>cLinOk</b>	Função executada para validar o contexto da linha atual do aCols
<b>cTudoOk</b>	Função executada para validar o contexto geral da MsNewGetDados (todo aCols)
<b>cIniCpos</b>	Nome dos campos do tipo caracter que utilizarão incremento automático.
<b>aAlterGDa</b>	Campos alteráveis da GetDados
<b>nFreeze</b>	Campos estáticos na GetDados, partindo sempre da posição inicial da getdados aonde:  1- Primeiro campo congelado 2- Primeiro e segundo campos congelados...
<b>nMax</b>	Número máximo de linhas permitidas. Valor padrão 99
<b>cFieldOk</b>	Função executada na validação do campo
<b>cSuperDel</b>	Função executada quando pressionada as teclas <Ctrl>+<Delete>
<b>cDelOk</b>	Função executada para validar a exclusão de uma linha do aCols
<b>oDLG</b>	Objeto no qual a MsNewGetDados será criada
<b>aHeader</b>	Array a ser tratado internamente na MsNewGetDados como aHeader
<b>aCols</b>	Array a ser tratado internamente na MsNewGetDados como aCols



**Aparência:**

Pedidos de Venda - Incluir

Cliente	<input type="text"/>	Loja	<input type="text"/>			
cli.Entrega	<input type="text"/>	Loja Entrega	<input type="text"/>			
Transp.	<input type="text"/>	Tipo Cliente	<input type="text"/>			
Cond. Pagto	<input type="text"/>	Tabela	<input type="text"/>			
Vendedor 1	<input type="text"/>	Comissao 1	0,00			
Vendedor 2	<input type="text"/>	Comissao 2	0,00			
Item	Produto	Unidade	Quantidade	Prc Unitario	Vlr Total	Qtd.Lib
			0,00	0,00	0,00	

**Variáveis private:**

<b>aRotina</b>	<p>Vetor com as rotinas que serão executadas na MBrowse e que definirá o tipo de operação que está sendo executada (inclusão, alteração, exclusão, visualização, pesquisa, ...) no formato:</p> <p>{cTítulo, cRotina, nOpção, nAcesso}, aonde:</p> <p>nOpção segue o padrão do ERP Protheus para:</p> <ul style="list-style-type: none"> <li>1- Pesquisar</li> <li>2- Visualizar</li> <li>3- Incluir</li> <li>4- Alterar</li> <li>5- Excluir</li> </ul>
<b>aHeader</b>	<p>Vetor com informações das colunas no formato:</p> <p>{cTítulo, cCampo, cPicture, nTamanho, nDecimais,; cValidação, cReservado, cTipo, xReservado1, xReservado2}</p> <p>A tabela temporária utilizada pela MsGetDB deverá ser criada com base no aHeader mais um último campo tipo lógico que determina se a linha foi excluída.</p>
<b>lRefresh</b>	Variável tipo lógica para uso reservado.

**Variáveis públicas:**

<b>N</b>	Indica qual a linha posicionada do aCols.
----------	---

**Funções de validação:**

<b>cLinhaOk</b>	Função de validação na mudança das linhas da grid. Não pode ser definida como Static Function.
<b>cTudoOk</b>	Função de validação da confirmação da operação com o grid. Não pode ser definida como Static Function.

**Métodos adicionais:**

<b>ForceRefresh()</b>	Atualiza a MsNewGetDados com a tabela e posiciona na primeira linha.
<b>Hide()</b>	Oculta a MsNewGetDados.
<b>Show()</b>	Mostra a MsNewGetDados.

## Exemplo: Utilização dos objetos MsNewGetDados() e MsMGet()

---

```
#include "protheus.ch"

/*
+-----
| Função      | MBRWGETD   | Autor | ARNALDO RAYMUNDO JR. | Data |
|             |            |       |                      |       |
+-----
| Descrição      | Programa que demonstra a utilização dos objetos|
|                 | MsNewGetDados() e MsMGet() combinados
|             |            |
+-----
| Uso          | Curso ADVPL
|             |
+-----
*/
User Function MrbwGetD()

Private cCadastro      := "Pedidos de Venda"
Private aRotina := [{"Pesquisar" , "axPesqui" , 0, 1}; {"Visualizar" , "U_ModGtd" , 0, 2}; {"Incluir" , "U_ModGtd" , 0, 3}]

DbSelectArea("SC5")
DbSetOrder(1)

MBrowse(6,1,22,75,"SC5")

Return

User Function ModGtd(cAlias,nReg,nOpc)

Local nX      := 0
Local nUsado  := 0
Local aButtons := {}
Local aCpoEnch := {}
Local cAliasE := cAlias
Local aAlterEnch := {}
Local aPos    := {000,000,080,400}
Local nModelo  := 3
Local lF3      := .F.
Local lMemoria := .T.
Local lColumn  := .F.
```

```

Local caTela           := ""
Local INoFolder := .F.
Local IProperty := .F.
Local aCpoGDa          := {}
Local cAliasGD         := "SC6"
Local nSuperior        := 081
Local nEsquerda        := 000
Local nInferior        := 250
Local nDireita         := 400
Local cLinOk            := "AllwaysTrue"
Local cTudoOk           := "AllwaysTrue"
Local cIniCpos          := "C6_ITEM"
Local nFreeze           := 000
Local nMax              := 999
Local cFieldOk          := "AllwaysTrue"
Local cSuperDel          := ""
Local cDelOk             := "AllwaysFalse"
Local aHeader            := {}
Local aCols              := {}
Local aAlterGDa := {}

Private oDlg
Private oGetD
Private oEnch
Private aTELA[0][0]
Private aGETS[0]

DbSelectArea("SX3")
DbSetOrder(1)
DbSeek(cAliasE)

While !Eof() .And. SX3->X3_ARQUIVO == cAliasE
    If !(SX3->X3_CAMPO $ "C5_FILIAL") .And. cNivel >= SX3->X3_NIVEL
    .And.:
        X3Uso(SX3->X3_USADO)
            AADD(aCpoEnch,SX3->X3_CAMPO)
            Endif
        DbSkip()
    End

    aAlterEnch := aClone(aCpoEnch)

DbSelectArea("SX3")
DbSetOrder(1)
MsSeek(cAliasGD)

While !Eof() .And. SX3->X3_ARQUIVO == cAliasGD
    If      !(AllTrim(SX3->X3_CAMPO) $ "C6_FILIAL") .And.:
        cNivel >= SX3->X3_NIVEL .And. X3Uso(SX3->X3_USADO)

```

```

        AADD(aCpoGDa,SX3->X3_CAMPO)
        EndIf
        DbSkip()
    End

    aAlterGDa := aClone(aCpoGDa)

    nUsado:=0
    dbSelectArea("SX3")
    dbSeek("SC6")

    aHeader:={}

    While !Eof().And.(x3_arquivo=="SC6")
        If X3USO(x3_usado).And.cNivel>=x3_nivel
            nUsado:=nUsado+1
            AADD(aHeader,{ TRIM(x3_titulo), x3_campo, x3_picture,x3_tamanho,;
            x3_decimal,"AllwaysTrue()",x3_usado, x3_tipo, x3_arquivo, x3_context
            })
        Endif
        dbSkip()
    End

    If nOpc==3 // Incluir
        aCols:={Array(nUsado+1)}
        aCols[1,nUsado+1]:=F.
        For nX:=1 to nUsado
            IF aHeader[nX,2] == "C6_ITEM"
                aCols[1,nX]:="0001"
            ELSE
                aCols[1,nX]:=CriaVar(aHeader[nX,2])
            ENDIF
        Next
    Else
        aCols:={}
        dbSelectArea("SC6")
        dbSetOrder(1)
        dbSeek(xFilial()+M->C5_NUM)
        While !eof().and.C6_NUM==M->C5_NUM
            AADD(aCols,Array(nUsado+1))
            For nX:=1 to nUsado

                aCols[Len(aCols),nX]:=FieldGet(FieldPos(aHeader[nX,2]))
            Next
            aCols[Len(aCols),nUsado+1]:=F.
            dbSkip()
        End
    Endif

    oDlg      := MSDIALOG():New(000,000,400,600, cCadastro,,,.T.)
    RegToMemory("SC5", If(nOpc==3,.T.,.F.))

```

```

oEnch :=
MsMGet():New(cAliasE,nReg,nOpc,/*aCRA*/,/*cLetra*/,/*cTexto*/,
               aCpoEnch,aPos,aAlterEnch, nModelo, /*nColMens*/,
/*cMensagem*/;;
/*cTudoOk*/, oDlg,IF3, IMemoria,IColumn,caTela,INoFolder,;
IProperty)

oGetD:= MsNewGetDados():New(nSuperior, nEsquerda, nInferior,
nDireita,;
nOpc,cLinOk,cTudoOk, cIniCpos, aAlterGDa, nFreeze,
nMax,cFieldOk,;
cSuperDel,cDelOk, oDLG, aHeader, aCols)

oDlg:bInit := {| | EnchoiceBar(oDlg,
{| | oDlg:End()},{| | oDlg:End(),,aButtons})
oDlg:ICentered := .T.
oDlg:Activate()
Return

```

### **2.32.3.1 Definindo cores personalizadas para o objeto MsNewGetDados()**

---

Conforme visto no tópico sobre definição das propriedades de cores para os componentes visuais, cada objeto possui características que devem ser respeitadas para correta utilização deste recurso.

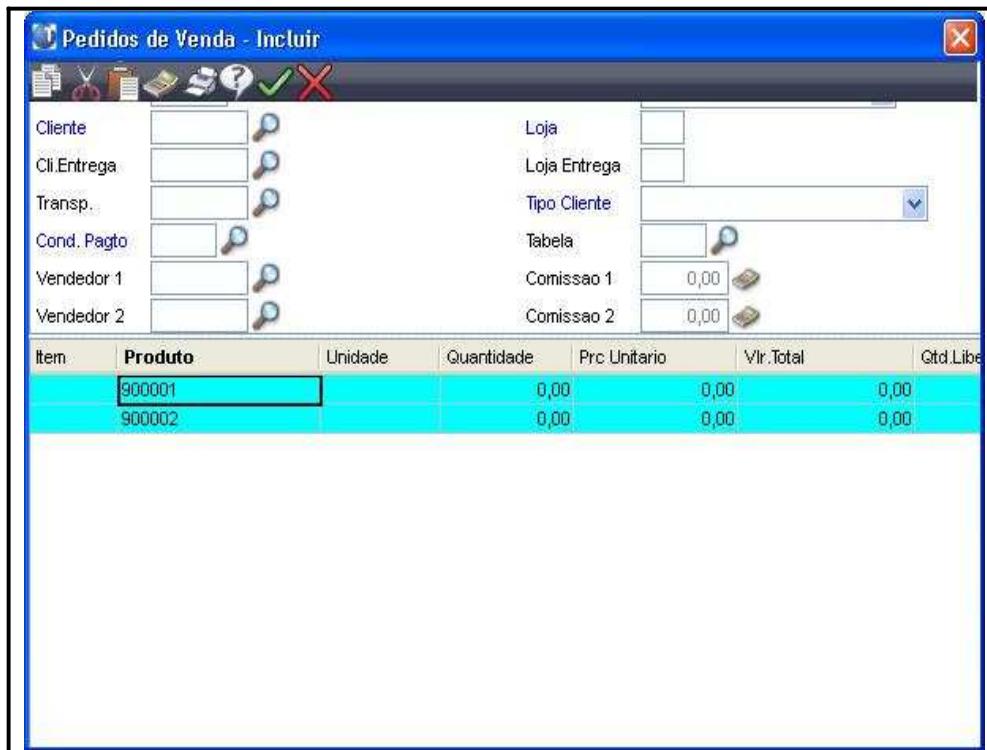
**Atributos adicionais:**

<b>IUseDefaultColors</b>	Atributo que deverá ser definido como .F. para que as alterações nas cores sejam permitidas.
--------------------------	--

**Métodos adicionais:**

<b>SetBlkBackColor</b>	Método que define a cor que será utilizada para cada linha do grid. Não é necessário utilizar o método Refresh() após a definição da cor por este método.
------------------------	---

### Aparência:



Exemplo: Definindo cores personalizadas para o objeto MsNewGetDados()

```
#include "protheus.ch"

/*
+-----
| Função      | MRBWTCL    | Autor | ARNALDO RAYMUNDO JR. | Data |
+-----
| Descrição      | Programa que demonstra a utilização dos objetos
|
|           | MsNewGetDados() e MsMGet() combinados e tratamento de
cores|
+-----
| Uso          | Curso ADVPL
|
+-----
*/
User Function MrbwGtCl()

Private cCadastro      := "Pedidos de Venda"
Private aRotina :=      {"Pesquisar" , "axPesqui" , 0, 1};
```

```

        {"Visualizar" , "U_ModGtd" , 0, 2},;
        {"Incluir" , "U_ModGtd" , 0, 3}}}

DbSelectArea("SC5")
DbSetOrder(1)

MBrowse(6,1,22,75,"SC5")

Return

User Function ModGtd(cAlias,nReg,nOpc)

Local nX      := 0
Local nUsado   := 0

Local aButtons    := {}
Local aCpoEnch   := {}
Local cAliasE    := cAlias
Local aAlterEnch := {}
Local aPos       := {000,000,080,400}
Local nModelo    := 3
Local lF3        := .F.
Local lMemoria   := .T.
Local lColumn    := .F.
Local caTela     := ""
Local lNoFolder  := .F.
Local lProperty  := .F.
Local aCpoGDa    := {}
Local cAliasGD   := "SC6"
Local nSuperior   := 081
Local nEsquerda  := 000
Local nInferior   := 250
Local nDireita    := 400
Local cLinOk      := "AllwaysTrue"
Local cTudoOk     := "AllwaysTrue"
Local cIniCpos   := "C6_ITEM"
Local nFreeze     := 000
Local nMax        := 999
Local cFieldOk   := "AllwaysTrue"
Local cSuperDel  := ""
Local cDelOk      := "AllwaysFalse"
Local aHeader     := {}
Local aCols       := {}
Local aAlterGDa  := {}

Private oDlg
Private oGetD
Private oEnch

```

```

Private aTEL[0][0]
Private aGETS[0]

DbSelectArea("SX3")
DbSetOrder(1)
DbSeek(cAliasE)

While !Eof() .And. SX3->X3_ARQUIVO == cAliasE
    If !(SX3->X3_CAMPO $ "C5_FILIAL") .And. cNivel >= SX3->X3_NIVEL
    .And.;

    X3Uso(SX3->X3_USADO)
        AADD(aCpoEnch,SX3->X3_CAMPO)
        EndIf
        DbSkip()
    End

    aAlterEnch := aClone(aCpoEnch)

DbSelectArea("SX3")
DbSetOrder(1)
MsSeek(cAliasGD)

While !Eof() .And. SX3->X3_ARQUIVO == cAliasGD
    If      !(AllTrim(SX3->X3_CAMPO) $ "C6_FILIAL") .And. cNivel >=
SX3->X3_NIVEL .And. X3Uso(SX3->X3_USADO)
        AADD(aCpoGDa,SX3->X3_CAMPO)
        EndIf
        DbSkip()
    End

    aAlterGDa := aClone(aCpoGDa)

nUsado:=0
dbSelectArea("SX3")
dbSeek("SC6")

aHeader:={}

While !Eof().And.(x3_arquivo=="SC6")
    If X3USO(x3_usado).And.cNivel>=x3_nivel
        nUsado:=nUsado+1
        AADD(aHeader,{ TRIM(x3_titulo), x3_campo, x3_picture,;
            x3_tamanho, x3_decimal,"AllwaysTrue()",;
            x3_usado, x3_tipo, x3_arquivo, x3_context } )
        Endif
        dbSkip()
    End

    If nOpc==3 // Incluir
        aCols:={Array(nUsado+1)}
        aCols[1,nUsado+1]:=F.

```

```

For nX:=1 to nUsado

    IF aHeader[nX,2] == "C6_ITEM"
        aCols[1,nX]:= "0001"
    ELSE
        aCols[1,nX]:=CriaVar(aHeader[nX,2])
    ENDIF

    Next

Else
    aCols:={}
    dbSelectArea("SC6")
    dbSetOrder(1)
    dbSeek(xFilial()+M->C5_NUM)
    While !eof().and.C6_NUM==M->C5_NUM
        AADD(aCols,Array(nUsado+1))
        For nX:=1 to nUsado

            aCols[Len(aCols),nX]:=FieldGet(FieldPos(aHeader[nX,2]))
            Next
            aCols[Len(aCols),nUsado+1]:=F.
            dbSkip()
        End
    Endif

oDlg := MSDIALOG():New(000,000,400,600, cCadastro,,,.T.)

    RegToMemory("SC5", If(nOpc==3.,T.,F.))

    oEnch := MsMGet():New(cAliasE,nReg,nOpc,/*aCRA*/,/*cLetra*/,
/*cTexto*/;;
                    aCpoEnch,aPos, aAlterEnch, nModelo, /*nColMens*/,
/*cMensagem*/;;
                    cTudoOk,oDlg,IF3,
IMemoria,IColumn,caTela,INoFolder,IProperty)

    oGetD:=
MsNewGetDados():New(nSuperior,nEsquerda,nInferior,nDireita, nOpc,;
                    cLinOk,cTudoOk,cIniCpos,aAlterGDa,nFreeze,nMax,cFieldOk,
cSuperDel,;
                    cDelOk, oDLG, aHeader, aCols)

// Tratamento para definição de cores específicas,
// logo após a declaração da MsNewGetDados

    oGetD:oBrowse:lUseDefaultColors := .F.
    oGetD:oBrowse:SetBlkBackColor({| |
GETCLR(oGetD:aCols,oGetD:nAt,aHeader)})
```

```

oDlg:bInit := {| | EnchoiceBar(oDlg, {| |oDlg:End()},
{| |oDlg:End()}},aButtons)}
oDlg:lCentered := .T.
oDlg:Activate()

Return

// Função para tratamento das regras de cores para a grid da
MsNewGetDados

Static Function GETDCLR(aLinha,nLinha,aHeader)

Local nCor2      := 16776960 // Ciano - RGB(0,255,255)
Local nCor3      := 16777215 // Branco - RGB(255,255,255)
Local nPosProd   := aScan(aHeader,{|x| Alltrim(x[2]) ==
"C6_PRODUTO"})
Local nUsado     := Len(aHeader)+1
Local nRet       := nCor3

If !Empty(aLinha[nLinha][nPosProd]) .AND. aLinha[nLinha][nUsado]
    nRet := nCor2
Elseif !Empty(aLinha[nLinha][nPosProd]) .AND.
!aLinha[nLinha][nUsado]
    nRet := nCor3
Endif

Return nRet

```

## 2.33 Barras de botões

A linguagem ADVPL permite a implementação de barras de botões utilizando funções pré-definidas desenvolvidas com o objetivo de facilitar sua utilização, ou através da utilização direta dos componentes visuais disponíveis. Dentre os recursos da linguagem que podem ser utilizados com esta finalidade serão abordados:

- . **Função EnchoiceBar:** Sintaxe tradicionalmente utilizada em ADVPL, a qual não retorna um objeto para a aplicação chamadora;
- . **Classe TBar:** Classe do objeto TBar(), a qual permite a instanciação direta de um objeto do tipo barra de botões superior, tornando-o disponível na aplicação chamadora.
- . **Classe ButtonBar:** Classe do objeto ButtonBar(), a qual permite a instanciação direta de um objeto barra de botões genérico, o qual pode ser utilizado em qualquer posição da tela, tornando-o disponível na aplicação chamadora.

## **2.33.1EnchoiceBar()**

Função que cria uma barra de botões no formato padrão utilizado pelas interfaces de cadastro da aplicação Protheus.

Esta barra possui os botões padrões para confirmar ou cancelar a interface e ainda permite a adição de botões adicionais com a utilização do parâmetro ***aButtons***.

**Sintaxe:**

<b>EnchoiceBar( oDlg, bOk, bCancel, lMsgDel, aButtons, nRecno, cAlias)</b>
--

**Parâmetros:**

<b>oDlg</b>	Dialog onde irá criar a barra de botões
<b>bOk</b>	Bloco de código a ser executado no botão Ok
<b>bCancel</b>	Bloco de código a ser executado no botão Cancelar
<b>lMsgDel</b>	Exibe dialog para confirmar a exclusão
<b>aButtons</b>	Array contendo botões adicionais. aArray[n][1] -> Imagem do botão aArray[n][2] -> bloco de código contendo a ação do botão aArray[n][3] -> título do botão
<b>nRecno</b>	Registro a ser posicionado após a execução do botão Ok.
<b>cAlias</b>	Alias do registro a ser posicionado após a execução do botão Ok. Se o parâmetro nRecno for informado, o cAlias passa ser obrigatório.

**Aparência:**



### **Exemplo: Utilização da função EnchoiceBar()**

---

```
#include "protheus.ch"

/*
+-----
| Função      | DENCHBAR   | Autor | ARNALDO RAYMUNDO JR. | Data |
|             |            |       |
+-----
| Descrição      | Programa que demonstra a utilização da função
|             |           |
|             | EnchoiceBar()
|             |
+-----
| Uso          | Curso ADVPL
|             |
+-----
*/
User Function DEnchBar()
Local oDlg, oBtn
Local aButtons := {}

DEFINE MSDIALOG oDlg TITLE "Teste EnchoiceBar" FROM 000,000 TO
400,600 PIXEL OF;
oMainWnd

AADD( aButtons, {"HISTORIC", {|| TestHist(), "Histórico...";}
               "Histórico", {|| .T.} } )

@ -15,-15 BUTTON oBtn PROMPT "..." SIZE 1,1 PIXEL OF oDlg

ACTIVATE MSDIALOG oDlg ;
ON INIT
(EnchoiceBar(oDlg,{|| IOk:=.T.,oDlg:End()},{|| oDlg:End()},@aButtons)
)

Return
```

### **2.33.2 TBar()**

Classe de objetos visuais que permite a implementação de um componente do tipo barra de botões para a parte superior de uma janela previamente definida.

**Sintaxe:** New(oWnd, nBtnWidth, nBtnHeight, l3D, cMode, oCursor, cResource, lNoAutoAdjust)

**Retorno:** oTBar – objeto do tipo TBar()

**Parâmetros:**

<b>oWnd</b>	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
<b>nBtnWidth</b>	Numérico, opcional. Largura do botão contido na barra
<b>nBtnHeight</b>	Numérico, opcional. Altura do botão contido na barra
<b>l3D</b>	Lógico, opcional. Define tipo da barra
<b>cMode</b>	Não utilizado.
<b>oCursor</b>	Objeto, opcional. Define Cursor ao posicionar o mouse sobre a barra.
<b>cResource</b>	Caracter, opcional. Imagem do recurso a ser inserido como fundo da barra.
<b>lNoAutoAdjust</b>	Lógico.

**Aparência:**



**Exemplo: Utilização da função EnchoiceBar()**

```
#include 'protheus.ch'

/*
+-----+
| Função      | TSTBAR          | Autor | MICROSIGA           | Data |
|             |                  |       |                      |
+-----+
-----+
| Descrição    | Programa que demonstra a utilização do objeto
TBar()|
+-----+
-----+
| Uso          | Curso ADVPL
|             |
+-----+
```

```

/*
User Function TstTBar()
Local oDlg

oDlg           := MSDIALOG():New(000,000,305,505, 'Exemplo -
TBAR' ,,,,.T.)

Exemplo (continuação):

oTBar := TBar():New( oDlg,25,32,.T.,,,.F. )

oTBtnBmp2_1      := TBtnBmp2():New( 00, 00, 35, 25, 'copyuser' ,,,;
{| |} |Alert('TBtnBmp2_1')}, oTBar,'msGetEx',,F.,.F. )

oTBtnBmp2_2      := TBtnBmp2():New( 00, 00, 35, 25, 'critica' ,,,;
{| |},oTBar,'Critica',,F.,.F. )

oTBtnBmp2_3      := TBtnBmp2():New( 00, 00, 35, 25, 'bmpcpo' ,,,;
{| |},oTBar,'PCO',,F.,.F. )

oTBtnBmp2_4      := TBtnBmp2():New( 00, 00, 35, 25, 'preco' ,,,;
{| |},oTBar,'Preço' ,,,F.,.F. )

oDlg:ICentered := .T.
oDlg:Activate()

Return

```

### 2.33.3 ButtonBar

A sintaxe **ButtonBar** é a forma clássica utilizada na linguagem ADVPL para implementar um objeto da classe TBar(), o qual possui as características mencionadas no tópico anterior.

#### Sintaxe:

<b>DEFINE BUTTONBAR oBar SIZE nWidth, nHeight 3D MODE OF oDlg CURSOR</b>
--

**Retorno: ()**.

#### Parâmetros:

<b>oBar</b>	Objeto do tipo TBar() que será criado com a utilização
-------------	--

	da sintaxe ButtonBar().
<b>nWidth</b>	Numérico, opcional. Largura do botão contido na barra.
<b>nHeight</b>	Numérico, opcional. Altura do botão contido na barra.
<b>3D</b>	Se definido habilita a visualização em 3D da barra de botões.
<b>oDlg</b>	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
<b>MODE</b>	Define a forma de orientação do objeto ButtonBar utilizando os seguintes termos pré-definidos:  TOP, BOTTOM, FLOAT
<b>CURSOR</b>	Objeto, opcional. Define Cursor ao posicionar o mouse sobre a barra.

A sintaxe ButtonBar requer a adição dos botões como recursos adicionais da barra previamente definida utilizando a sintaxe abaixo:

#### Botões: BUTTON RESOURCE

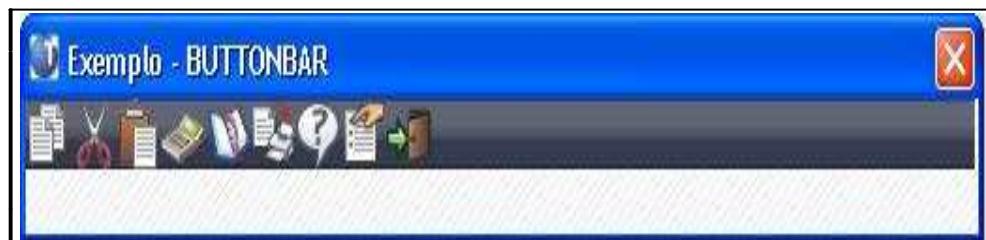
##### Sintaxe adicional:

```
DEFINE BUTTON RESOURCE cBitMap          OF oBar ACTION cAcao
TOOLTIP cTexto
```

##### Parâmetros:

<b>cBitMap</b>	Nome da imagem disponível na aplicação.
<b>oBar</b>	Objeto do tipo TBar() no qual o botão será adicionado.
<b>cAcao</b>	Função ou lista de expressões que determina a ação que será realizada pelo botão.
<b>cTexto</b>	Texto no estilo “tooltip text” que será exibido quando o cursor do mouse for posicionado sobre o botão na barra de ferramentas.

##### Aparência:



## Exemplo: Utilização da sintaxe ButtonBar

---

```
#include 'protheus.ch'

/*
+-----
---| Função      | TstBBar          | Autor | MICROSIGA           | Data |
|           |                 |       |                      |
+-----|-----|-----|-----|-----|-----|
---| Descrição    | Programa que demonstra a utilização do objeto
TBar()|           |
+-----|-----|-----|-----|-----|
---| Uso          | Curso ADVPL
|           |                 |
+-----|-----|-----|-----|-----|
---*/
User Function TstBBar()

Local oDlg
Local oBtn1
Local oBtn2

oDlg      := MSDIALOG():New(000,000,305,505, 'Exemplo -
BUTTONBAR' ,,,,.T.)

DEFINE BUTTONBAR oBar SIZE 25,25 3D TOP OF oDlg

DEFINE BUTTON RESOURCE "S4WB005N"  OF oBar ACTION NaoDisp() TOOLTIP
"Recortar"
DEFINE BUTTON RESOURCE "S4WB006N"  OF oBar ACTION NaoDisp() TOOLTIP
"Copiar"
DEFINE BUTTON RESOURCE "S4WB007N"  OF oBar ACTION NaoDisp() TOOLTIP
"Colar"
DEFINE BUTTON oBtn1 RESOURCE "S4WB008N" OF oBar GROUP;
ACTION Calculadora() TOOLTIP "Calculadora"

oBtn1:cTitle:="Calc"
DEFINE BUTTON RESOURCE "S4WB009N"  OF oBar ACTION Agenda() TOOLTIP
"Agenda"
DEFINE BUTTON RESOURCE "S4WB010N"  OF oBar ACTION OurSpool() TOOLTIP
"Spool"
DEFINE BUTTON RESOURCE "S4WB016N"  OF oBar GROUP;
ACTION HelProg() TOOLTIP "Ajuda"
```

```
DEFINE BUTTON oBtn2 RESOURCE "PARAMETROS" OF oBar GROUP;
ACTION Sx1C020()  TOOLTIP "Parâmetros"

oBtn2:cTitle:="Param."

DEFINE      BUTTON oBtOk RESOURCE "FINAL" OF oBar GROUP;
ACTION oDlg:End()TOOLTIP "Sair"

oBar:bRClicked := {} | AllwaysTrue()
oDlg:lCentered := .T.
oDlg:Activate()

Return
```

## **2.33.4 Imagens pré-definidas para as barras de botões**

Conforme mencionado nos tópicos anteriores, os botões visuais do tipo barra de botões permitem a definição de itens com ações e imagens vinculadas.

Dentre os objetos e funções mencionados, foi citada a EnchoiceBar(), a qual permite a adição de botões adicionais através do parâmetro **aButton**, sendo que os itens deste array devem possuir o seguinte formato:

**Sintaxe:** AADD(aButtons,{cBitMap, bAcao, cTexto})

**Estrutura:**

<b>cBitMap</b>	Nome da imagem pré-definida existente na aplicação ERP que será vinculada ao botão.
<b>bAcao</b>	Bloco de código que define a ação que será executada com a utilização do botão.
<b>cTexto</b>	Texto no estilo “tooltip text” que será exibido quando o cursor do mouse for posicionado sobre o botão na barra de ferramentas.

**Alguns BitMaps disponíveis:**



**Exemplo:**

```
AADD(aButtons,{"USER",{| | AllwaysTrue(),"Usuário"})
```

**APÊNDICE**

**BOAS PRÁTICAS DE PROGRAMAÇÃO**

## **16. Arredondamento**

Algumas operações numéricas podem causar diferenças de arredondamento. Isso ocorre devido a diferenças no armazenamento de variáveis numéricas nos diversos processadores, diferença esta, inclusive, presente no ADVPL, mesmo antes do surgimento do Protheus.

Para evitar esses problemas de arredondamento, deve ser utilizada a função Round(), principalmente antes de realizar uma comparação e antes de se utilizar a função Int().

Desse modo, assegura-se que o resultado será correto independentemente do processador ou plataforma.

**Exemplo 01:**

```
If (Valor/30) == 50          // pode ser falso ou inválido  
If Round(Valor/30, 0) == 50 // correto
```

**Exemplo 02:**

```
M->EE8_QTDEM1 := Int(M->EE8_SLDINI/M->EE8_QE) // pode ser falso ou  
inválido  
M->EE8_QTDEM1 := Int(Round(M->EE8_SLDINI/M->EE8_QE,10)) // correto
```

## 17. Utilização de Identação

É obrigatória a utilização da identação, pois torna o código muito mais legível. Veja os exemplos abaixo:

```
While !SB1->(Eof())
If mv_par01 == SB1->B1_COD
dbSkip()
Loop
Endif
Do Case
Case SB1->B1_LOCAL == "01" .OR. SB1->B1_LOCAL == "02"
TrataLocal(SB1->B1_COD, SB1->B1_LOCAL)
Case SB1->B1_LOCAL == "03"
TrataDefeito(SB1->B1_COD)
OtherWise
TrataCompra(SB1->B1_COD, SB1->B1_LOCAL)
EndCase
dbSkip()
EndDo
```

A utilização da identação seguindo as estruturas de controle de fluxo (while, IF, caso etc.) torna a compreensão do código muito mais fácil:

```
While !SB1->(Eof())

    If mv_par01 == SB1->B1_COD
        dbSkip()
        Loop
    Endif

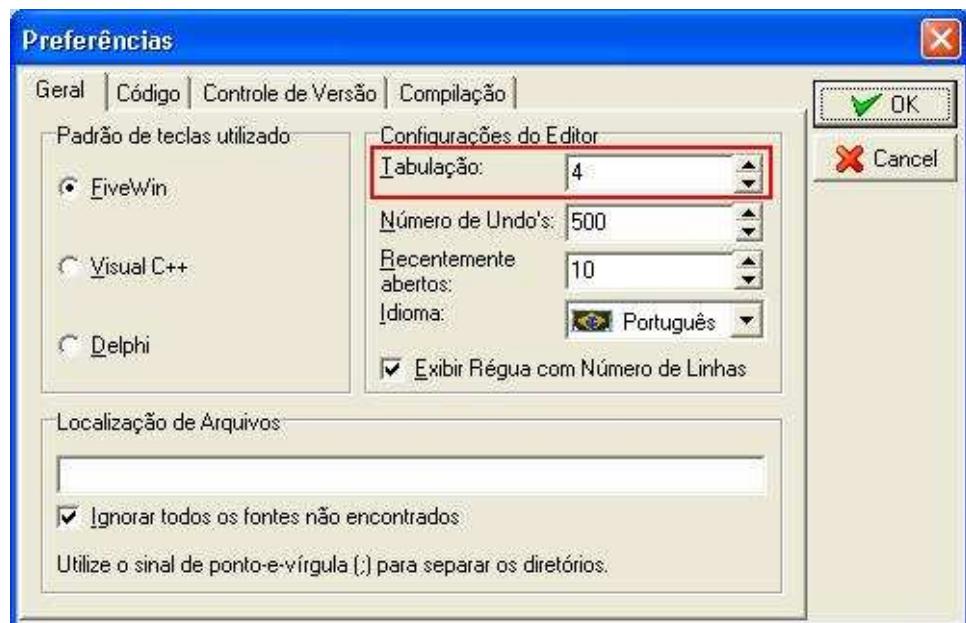
    Do Case
        Case SB1->B1_LOCAL == "01" .OR. SB1->B1_LOCAL == "02"
            TrataLocal(SB1->B1_COD, SB1->B1_LOCAL)

        Case SB1->B1_LOCAL == "03"
            TrataDefeito(SB1->B1_COD)

        OtherWise
            TrataCompra(SB1->B1_COD, SB1->B1_LOCAL)
    EndCase
    dbSkip()

EndDo
```

Para identar o código utilize a tecla <TAB> e na ferramenta DEV-Studio, a qual pode ser configurada através da opção “Preferências”:



## 18. Capitulação de Palavras-Chave

Uma convenção amplamente utilizada é a de capitular as palavras chaves, funções, variáveis e campos utilizando uma combinação de caracteres em maiúsculo e minúsculo, visando facilitar a leitura do código fonte. O código a seguir:

```
Local ncnt while ( ncnt++ < 10 ) ntotal += ncnt * 2 enddo
```

Ficaria melhor com as palavras chaves e variáveis capituladas:

```
Local nCnt While ( nCnt++ < 10 ) nTotal += nCnt * 2 EndDo
```

---

Para funções de manipulação de dados que comecem por “db”, a capitulação só será efetuada após o “db”:



*Importante*

- . dbSeek()
  - . dbSelectArea()
-

## 2.34 Palavras em maiúsculo

A regra é utilizar caracteres em maiúsculo para:

**Constantes:**

```
#define NUMLINES 60 #define NUMPAGES 1000
```

**Variáveis de memória:**

```
M-> CT2_CRCNV M->CT2_MCONVER := CriaVar("CT2_CONVERT")
```

**Campos:**

```
SC6->C6_NUMPED
```

**Querys:**

```
SELECT * FROM...
```

## 19. Utilização da Notação Húngara

A notação húngara consiste em adicionar os prefixos aos nomes de variáveis, de modo a facilmente se identificar seu tipo. Isto facilita na criação de códigos-fonte extensos, pois usando a Notação Húngara, você não precisa ficar o tempo todo voltando à definição de uma variável para se lembrar qual é o tipo de dados que deve ser colocado nela. Variáveis devem ter um prefixo de Notação Húngara em minúsculas, seguido de um nome que identifique a função da variável, sendo que a inicial de cada palavra deve ser maiúscula.

É obrigatória a utilização desta notação para nomear variáveis.

Notação	Tipo de dado	Exemplo
a	Array	aValores
b	Bloco de código	bSeek
c	Caracter	cNome
d	Data	dDataBase
l	Lógico	lContinua
n	Numérico	nValor
o	Objeto	oMainWindow
x	Indefinido	xConteudo

## 20. Técnicas de programação eficiente

Para o desenvolvimento de sistemas e a programação de rotinas, sempre é esperado que qualquer código escrito seja:

- ❑ **Funcionalmente correto**
- ❑ **Eficiente**
- ❑ **Legível**
- ❑ **Reutilizável**
- ❑ **Extensível**
- ❑ **Portável**

Após anos de experiência na utilização de linguagens padrão xBase e do desenvolvimento da linguagem ADVPL, algumas técnicas para uma programação otimizada e eficiente foram reconhecidas. A utilização das técnicas a seguir, visa buscar o máximo aproveitamento dos recursos da linguagem com o objetivo de criar programas com estas características.

## Criação de funções segundo a necessidade

Observe o código de exemplo:

```
User Function GetAnswer(IDefault)
Local IOk
IOk := GetOk(IDefault)
If IOk
    Return .T.
Else
    Return .F.
Endif
Return nil
```

Utilizando-se apenas o critério "a função funciona corretamente?", a função GetAnswer é perfeita. Recebe um parâmetro lógico com a resposta padrão e retorna um valor lógico dependente da opção escolhida pelo usuário em uma função de diálogo "sim/não" designada para isso. Pode entretanto ser melhorada, particularmente se eficiência for considerada como um critério para um código melhor. Eficiência tipicamente envolve a utilização de poucos recursos de máquina, poucos chamadas de funções ou tornar mais rápido um processo.

Segundo esse raciocínio, poderia se produzir o seguinte código:

```
User Function GetAnswer(IDefault)
Return If( GetOk(IDefault), .T., .F.)
```

O código acima ainda pode ser aprimorado conforme abaixo:

```
User Function GetAnswer(IDefault)
Return GetOk(IDefault)
```

Com a otimização do código da função GetAnswer(), pode facilmente verificar que a mesma não realiza nada adicional à chamada de GetOk(), podendo ser substituída por uma chamada direta desta, continuando a funcionar corretamente.

## Codificação auto-documentável

Nenhum comentário substitui um código claramente escrito, e este não é um acidente. Considere o exemplo:

```
cVar := "           " // 11 espaços
```

O tamanho da variável cVar não é evidente por si só e não é facilmente verificado. Estes mesmos 10 espaços estariam mais óbvios e ainda assim garantidos se a instrução fosse escrita como:

```
cVar := Space(11)
```

O mesmo princípio pode ser aplicado para qualquer string longa de caracteres repetidos. A função Replicate pode ser utilizada como a seguir:

```
cVar := Replicate( "*", 80 )
```

Este tipo de programação deixa o código fácil de digitar, fácil de ler e mais flexível.

## Utilização de soluções simples

---

Simplicidade na criação de instruções torna a programação e até mesmo a execução mais rápida. Considere a linha de código:

```
If nVar > 0 .Or. nVar < 0
```

Se o valor da variável nVar for igual a zero (0) no momento da execução desta linha de código, ambas as comparações separadas pelo operador lógico .Or. serão efetuadas: Após ser avaliada, a primeira comparação irá falhar. A segunda comparação será então avaliada e falhará também. Como resultado, o código existente dentro da estrutura de fluxo If não será executado. Tal código somente será executado quando o valor desta variável for maior OU menor do que zero. Ou seja, sempre que for DIFERENTE de zero, o que torna a linha a seguir mais eficiente:

```
If nVar != 0
```

Este tipo de alteração torna o código mais legível e o processamento mais rápido, evitando a avaliação de instruções desnecessariamente.

Existem outras situações onde a simplificação pode ser utilizada. A expressão de avaliação a seguir:

```
If cVar == "A" .Or. cVar == "B" .Or. cVar == "C" .Or. cVar == "D"
```

Pode ser substituído pelo operador de contenção:

```
If cVar $ "ABCD"
```

### **Opção por flexibilidade**

A melhor solução é aquela que envolve o problema imediato e previne problemas no futuro. Considere o exemplo:

```
@nRow,nCol PSAY cVar Picture "!!!!!!!!!!!!!!!"
```

Exceto contando-se os caracteres, não existe maneira de saber se o número de caracteres de exclamação é o esperado. Enquanto isto é um problema, existem algo mais grave. A expressão de picture é estática. Se no futuro for necessário ajustar o tamanho da variável cVar, será necessário localizar todos os lugares no código onde esta máscara de picture está sendo utilizada para ajuste manual.

Existe uma opção de solução de auto-ajuste disponível que é fácil de digitar e tem a garantia de executar a tarefa igualmente (tornar todos os caracteres maiúsculos):

```
@nRow,nCol PSAY cVar Picture "@!"
```

## **Opção da praticidade ao drama**

---

Se a solução parece complexa, provavelmente é porque o caminho escolhido está levando a isso. Deve-se sempre se perguntar porque alguém desenvolveria uma linguagem que requisite tantos comandos complicados para fazer algo simples. Na grande maioria dos casos, existe uma solução mais simples. O exemplo abaixo deixa isso bem claro:

```
@ 10,25 Say Substr(cCep,1,5) + "-" + Substr(cCep,6,3) Picture  
"!!!!!!!"
```

Este código pode ser escrito de uma forma muito mais simples, conforme demonstrado abaixo:

```
@ 10,25 Say cCep Picture "@R 99999-999"
```

## **Utilização de operadores de incremento/decremento**

---

Utilizados devidamente, os operadores de incremento e decremento tornam o código mais fácil de ler e possivelmente um pouco mais rápidos. Ao contrário de escrever adições simples como:

```
nVar := nVar + 1  
nVar := nVar -1
```

Pode-se escrevê-las assim:

```
++nVar  
--nVar
```

Deve-se apenas tomar cuidado com a precedência destes operadores, pois o "++" ou o "--" podem aparecer antes ou depois de uma variável, e em alguns casos quando a variável for utilizada dentro de uma expressão, a prefixação ou sufixação destes operadores afetará o resultado. Para maiores detalhes, consulte a documentação de operadores da linguagem ADVPL.

## Evitar passos desnecessários

---

Existe uma diferença entre um bom hábito e perda de tempo. Algumas vezes estes conceitos podem estar muito próximos, mas um modo de diferenciá-los é balancear os benefícios de realizar alguma ação contra o problema que resultaria se não fosse executada. Observe o exemplo:

```
Local nCnt := 0
For nCnt := 1 To 1
< código >
Next nCnt
```

Inicializar a variável no momento da declaração não é um problema. Se o 0 fosse necessário no exemplo, teria sido útil a inicialização na declaração. Mas neste caso a estrutura de repetição **For...Next** atribui o seu valor imediatamente com 1, portanto não houve ganho em atribuir a variável com 0 no começo.

Neste exemplo não há nenhum ponto negativo e nada errado ocorrerá se a variável não for inicializada, portanto é aconselhável evitar este tipo de inicialização, pois não torna o código mais seguro e também não expressa a intenção do código mais claramente.

Porém, note este exemplo, onde a variável não é inicializada:

```
Local nCnt
While ( nCnt++ < 10 )
< código >
EndDo
```

Em ADVPL, variáveis não inicializadas sempre tem seu valor contendo nulo (nil) a princípio, o que fará com que uma exceção em tempo de execução aconteça quando a instrução de repetição while for executada.

Diferentemente do primeiro exemplo, onde a inicialização da variável não fazia diferença alguma, neste segundo exemplo a inicialização é absolutamente necessária. Deve-se procurar inicializar variáveis numéricas com zero (0) e variáveis caracter com string nula ("") apenas quando realmente necessário.

## Utilização de alternativas

---

Quando se está trabalhando em uma simples rotina, deve-se tomar algum tempo para explorar duas ou três diferentes abordagens. Quando se está trabalhando em algo mais complexo, deve-se planejar prototipar algumas a mais. Considere o seguinte código:

```
If cHair = "A"
    Replace hair With "Loira"
Else
    If cHair = "B"
        Replace hair With "Morena"
    Else
        If cHair = "C"
            Replace hair With "Ruiva"
        Else
            If cHair = "D"
                Replace hair With "Grisalho"
            Else
                Replace hair With "Preto"
            Endif
        Endif
    Endif
Endif
```

Um código de uma única letra, (A até E), foi informado para indicar a cor de cabelo. Este código foi então convertido e armazenado como uma string. Pode-se notar que a cor "Preto" será atribuída se nenhuma outra opção for verdadeira.

Uma alternativa que reduz o nível de identação torna o código mais fácil de ler enquanto reduz o número de comandos replace:

```
Do Case
    Case cHair == "A"
        cColor := "Loira"
    Case cHair == "B"
        cColor := "Morena"
    Case cHair == "C"
        cColor := "Ruiva"
    Case cHair == "D"
        cColor := "Grisalho"
    OtherWise
        cColor := "Preto"
EndCase

Replace hair With cColor
```

## **Utilização de arquivos de cabeçalho quando necessário**

---

Se um arquivo de código criado se referencia a comandos para interpretação e tratamento de arquivos XML, este deve se incluir o arquivo de cabeçalho próprio para tais comandos (XMLXFUN.CH no exemplo). Porém não deve-se incluir arquivos de cabeçalho apenas por segurança. Se não se está referenciando nenhuma das constantes ou utilizando nenhum dos comandos contidos em um destes arquivos, a inclusão apenas tornará a compilação mais demorada.

## **Constantes em maiúsculo**

---

Isto é uma convenção que faz sentido. Em ADVPL, como em C por exemplo, a regra é utilizar todos os caracteres de uma constante em maiúsculo, a fim de que possam ser claramente reconhecidos como constantes no código, e que não seja necessários lembrar onde foram declarados.

## **Utilização de identação**

---

Este é um hábito que todo programador deve desenvolver. Não consome muito esforço para manter o código alinhado durante o trabalho, porém quando necessário pode-se utilizar a ferramenta TOTVS DevStudio para a re-identação de código. Para maiores detalhes, consulte a documentação sobre a identação de códigos fontes disponível nos demais tópicos deste material.

## **Utilização de espaços em branco**

---

Espaços em branco extras tornam o código mais fácil para a leitura. Não é necessário imensas áreas em branco, mas agrupar pedaços de código através da utilização de espaços em branco funciona muito bem. Costuma-se separar parâmetros com espaços em branco.

## **Quebra de linhas muito longas**

---

Com o objetivo de tornar o código mais fácil de ler e imprimir, as linhas do código não devem estender o limite da tela ou do papel. Podem ser "quebradas" em mais de uma linha de texto utilizando o ponto-e-vírgula (;).

## **Capitulação de palavras-chave**

---

Uma convenção amplamente utilizada é a de capitular as palavras chaves, funções, variáveis e campos utilizando uma combinação de caracteres em maiúsculo e minúsculo, visando facilitar a leitura do código fonte.

Avaliando o código a seguir:

```
local ncnt
while ( ncnt++ < 10 )
ntotal += ncnt * 2
enddo
```

O mesmo ficaria muito mais claro se re-escrito conforme abaixo:

```
Local nCnt
While ( nCnt++ < 10 )
    nTotal += nCnt * 2
EndDo
```

## **Utilização da Notação Húngara**

---

A Notação Húngara é muito comum entre programadores xBase e de outras linguagens. A documentação do ADVPL utiliza esta notação para a descrição das funções e comandos e é aconselhável sua utilização na criação de rotinas, pois ajuda a evitar pequenos erros e facilita a leitura do código. Para maiores detalhes, consulte a documentação sobre a utilização da Notação Húngara de códigos fontes disponível nos demais tópicos deste material.

## **Utilização de nomes significantes para variáveis**

---

A principal vantagem da liberdade na criação dos nomes de variáveis é a facilidade de identificação da sua utilidade. Portanto deve-se utilizar essa facilidade o máximo possível. Nomes sem sentido apenas tornarão difícil a identificação da utilidade de uma determinada variável, assim como nomes extremamente curtos. Nem sempre a utilização de uma variável chamada *i* é a melhor saída. Claro, não convém criar uma variável com um nome muito longo que será utilizada como um contador, e referenciada muitas vezes no código. O bom senso deve ser utilizado.

Criar variáveis como *nNúmero* ou *dData* também não ajudam na identificação. A Notação Húngara já está sendo utilizada para isso e o objetivo do nome da variável deveria ser identificar sua utilização, não o tipo de dado utilizado. Deve-se procurar substituir tais variáveis por algo como *nTotal* ou *dCompra*.

O mesmo é válido para nomes de funções, que devem descrever um pouco sobre o que a função faz. Novamente nomes extremamente curtos não são aconselháveis.

## **Utilização de comentários**

---

Comentários são muito úteis na documentação de programas criados e para facilitar a identificação de processos importantes no futuro e devem sempre ser utilizados.

Sempre que possível, funções criadas devem ter uma breve descrição do seu objetivo, parâmetros e retorno. Além de servir como documentação, os comentários embelezam o código ao separar as funções umas das outras. Os comentários devem ser utilizados com bom senso, pois reescrever a sintaxe ADVPL em português torna-se apenas perda de tempo:

```
If nLastKey == 27 // Se o nLastKey for igual a 27
```

## **Criação de mensagens sistêmicas significantes e consistentes**

Seja oferecendo assistência, exibindo mensagens de aviso ou mantendo o usuário informado do estado de algum processo, as mensagens devem refletir o tom geral e a importância da aplicação. Em termos gerais, deve-se evitar ser muito informal e ao mesmo tempo muito técnico.

```
"Aguarde. Reindexando (FILIAL+COD+ LOCAL) do arquivo:  
\DADOSADV\SB1990.DBF"
```

Esse tipo de mensagem pode dar informações demais para o usuário e deixá-lo sentindo-se desconfortável se não souber o que significa "reindexando", etc. E de fato, o usuário não devia ser incomodado com tais detalhes. Apenas a frase "Aguarde, indexando." funcionaria corretamente, assim como palavras "processando" ou "reorganizando".

Outra boa idéia é evitar a referência a um item corrente de uma tabela como um "registro":

```
"Deletar este registro?"
```

Se a operação estiver sendo efetuada em um arquivo de clientes, o usuário deve ser questionado sobre a remoção do cliente corrente, se possível informando valores de identificação como o código ou o nome.

## **Evitar abreviação de comandos em 4 letras**

---

Apesar do ADVPL suportar a abreviação de comandos em quatro letras (por exemplo, repl no lugar de replace) não há necessidade de utilizar tal funcionalidade. Isto apenas torna o código mais difícil de ler e não torna a compilação mais rápida ou simples.

### **Evitar "disfarces" no código**

---

Não deve-se criar constantes para expressões complexas. Isto tornará o código muito difícil de compreender e poderá causar erros primários, pois pode-se imaginar que uma atribuição é efetuada a uma variável quando na verdade há toda uma expressão disfarçada:

```
#define NUMLINES aPrintDefs[1]
#define NUMPAGES aPrintDefs[2]
#define ISDISK    aReturn[5]

If ISDISK == 1
    NUMLINES := 55
Endif

NUMPAGES += 1
```

A impressão que se tem após uma leitura deste código é de que valores estão sendo atribuídos às variáveis ou que constantes estão sendo utilizadas. Se o objetivo é flexibilidade, o código anterior deve ser substituído por:

```
#define NUMLINES 1
#define NUMPAGES 2
#define ISDISK    5

If aReturn[ISDISK] == 1
    aPrintDefs[ NUMLINES ] := 55
Endif

aPrintDefs[ NUMPAGES ] += 1
```

### **Evitar código de segurança desnecessário**

---

Dada sua natureza binária, tudo pode ou não acontecer dentro de um computador. Adicionar pedaços de código apenas para "garantir a segurança" é freqüentemente utilizado como uma desculpa para evitar corrigir o problema real. Isto pode incluir a checagem para validar intervalos de datas ou para tipos de dados corretos, o que é comumente utilizando em funções:

```
Static Function MultMalor( nVal )
```

```
If ValType( nVal ) != "N"  
    nVal := 0  
Endif  
Return ( nVal * nVal )
```

O ganho é irrisório na checagem do tipo de dado do parâmetro já que nenhum programa corretamente escrito em execução poderia enviar uma string ou uma data para a função. De fato, este tipo de "captura" é o que torna a depuração difícil, já que o retorno será sempre um valor válido (mesmo que o parâmetro recebido seja de tipo de dado incorreto). Se esta captura não tiver sido efetuada quando um possível erro de tipo de dado inválido ocorrer, o código pode ser corrigido para que este erro não mais aconteça.

### **Isolamento de strings de texto**

---

No caso de mensagens e strings de texto, a centralização é um bom negócio. Pode-se colocar mensagens, caminhos para arquivos, e mesmo outros valores em um local específico. Isto os torna acessíveis de qualquer lugar no programa e fáceis de gerenciar.

Por exemplo, se existe uma mensagem comum como "***Imprimindo, por favor  
aguarde...***" em muitas partes do código, corre-se o risco de não seguir um padrão para uma das mensagens em algum lugar do código. E mantê-las em um único lugar, como um arquivo de cabeçalho, torna fácil a produção de documentação e a internacionalização em outros idiomas.

# **GUIA DE REFERÊNCIA RÁPIDA: FUNÇÕES E COMANDOS ADVPL**

Neste guia de referência rápida serão descritas as funções básicas da linguagem ADVPL, incluindo as funções herdadas da linguagem Clipper, necessárias ao desenvolvimento no ambiente ERP.

## **Conversão entre tipos de dados**

### **CTOD()**

Realiza a conversão de uma informação do tipo caracter no formato “DD/MM/AAAA” para uma variável do tipo data.

**Sintaxe: CTOD(cData)**

**Parâmetros**

<b>cData</b>	Caracter no formato “DD/MM/AAAA”
--------------	----------------------------------

**Exemplo:**

```
cData := "31/12/2006"  
dData := CTOD(cData)  
  
IF dDataBase >= dData  
    MSGALERT("Data do sistema fora da competência")  
ELSE  
    MSGINFO("Data do sistema dentro da competência")  
ENDIF
```

### **CVALTOCHAR()**

Realiza a conversão de uma informação do tipo numérico em uma *string*, sem a adição de espaços a informação.

**Sintaxe: CVALTOCHAR(nValor)**

**Parâmetros**

<b>nValor</b>	Valor numérico que será convertido para caractere.
---------------	--

**Exemplo:**

```
FOR nPercorridos := 1 to 10
    MSGINFO("Passos percorridos: "+CvalToChar(nPercorridos))
NEXT nPercorridos
```

---

**DTOC()**

Realiza a conversão de uma informação do tipo *data* para caracter, sendo o resultado no formato “DD/MM/AAAA”.

**Sintaxe: DTOC(dData)**

**Parâmetros**

<b>dData</b>	Variável com conteúdo data
--------------	----------------------------

**Exemplo:**

```
MSGINFO("Database do sistema: "+DTOC(dData))
```

---

**DTOS()**

Realiza a conversão de uma informação do tipo data em um caracter, sendo o resultado no formato “AAAAMMDD”.

**Sintaxe: DTOS(dData)**

**Parâmetros**

<b>dData</b>	Variável com conteúdo data
--------------	----------------------------

**Exemplo:**

```
cQuery := "SELECT A1_COD, A1_LOJA, A1_NREDUZ FROM SA1010 WHERE "
cQuery += "A1_DULTCOM >=""+DTOS(dDataIni)+""
```

---

**STOD()**

Realiza a conversão de uma informação do tipo caracter com conteúdo no formato “AAAAMMDD” em data.

**Sintaxe: STOD(sData)**

**Parâmetros**

<b>sData</b>	<i>String</i> no formato “AAAAMMDD”
--------------	-------------------------------------

**Exemplo:**

```
sData := LERSTR(01,08) // Função que realiza a leitura de uma          string
de um txt previamente
// aberto
dData := STOD(sData)
```

---

## STR()

Realiza a conversão de uma informação do tipo numérico em uma *string*, adicionando espaços à direita.

- Sintaxe: STR(*nValor*)**

- Parâmetros**

<b>nValor</b>	Valor numérico que será convertido para caractere.
---------------	--

**Exemplo:**

```
FOR nPercorridos := 1 to 10
  MSGINFO("Passos percorridos: "+CvalToChar(nPercorridos))
NEXT nPercorridos
```

---

## STRZERO()

Realiza a conversão de uma informação do tipo numérico em uma *string*, adicionando zeros à esquerda do número convertido, de forma que a *string* gerada tenha o tamanho especificado no parâmetro.

- Sintaxe: STRZERO(*nValor*, *nTamanho*)**

- Parâmetros**

<b>nValor</b>	Valor numérico que será convertido para caractere.
<b>nTamanho</b>	Tamanho total desejado para a <i>string</i> retornada.

**Exemplo:**

```
FOR nPercorridos := 1 to 10
  MSGINFO("Passos percorridos: "+CvalToChar(nPercorridos))
NEXT nPercorridos
```

## **VAL()**

---

Realiza a conversão de uma informação do tipo caracter em numérica.

. . . **Sintaxe:** **VAL(cValor)**

. . . **Parâmetros**

<b>cValor</b>	<i>String que será convertida para numérico.</i>
---------------	--

**Exemplo:**

```
Static Function Modulo11(cData)
LOCAL L, D, P := 0
L := Len(cdata)
D := 0
P := 1
While L > 0
    P := P + 1
    D := D + (Val(SubStr(cData, L, 1)) * P)
    If P = 9
        P := 1
    End
    L := L - 1
End
D := 11 - (mod(D,11))
If (D == 0 .Or. D == 1 .Or. D == 10 .Or. D == 11)
    D := 1
End
Return(D)
```

## **Matemáticas**

### **ACOS()**

Função utilizada para calcular o valor do arco co-seno.

. . . **Sintaxe:** **ACOS(nValor)**

. . . **Parâmetros:**

<b>nValor</b>	Valor entre -1 e 1 de quem será calculado o Arco Co-Seno.
---------------	---

**Retorno:**

<b>Numérico</b>	Range de 0 a $\pi$ radianos.  Se o valor informado no parâmetro for menor que -1 ou maior que 1, retorna um valor indefinido por default $[+\infty, -\infty]$	<b>acos</b>
-----------------	---	-------------

**CEILING()**

---

Função utilizada para calcular o valor mais próximo possível de um valor nMax informado como parâmetro para a função.

**Sintaxe: CELLING(nMax)****Parâmetros**

<b>nMax</b>	Valor limite para análise da função, no formato floating-point.
-------------	---

**Retorno:**

<b>Numérico</b>	Valor do tipo <i>double</i> , representando o menor inteiro que é maior ou igual ao valor de nX. Não há retorno de erro na função.
-----------------	--

**COS()**

---

Função utilizada para calcular o valor do co-seno ou co-seno hiperbólico.

**Importante:** Se  $x \geq 2^{63}$  ou  $x \leq -2^{63}$  ocorre perda significante na chamada da função COS().

**Sintaxe: COS(nAngulo)****Parâmetros:**

<b>nAngulo</b>	Valor que representa o ângulo em radianos.
----------------	--

**Retorno:**

<b>Numérico</b>	Valor que representa o co-seno ou co-seno hiperbólico do ângulo informado.
-----------------	--

### Situações inválidas:

Entrada	Exceção apresentada	Significado da Exceção
$\pm \text{QNAN}, \text{IND}$	None	Sem Domínio
$\pm \infty (\cosf, \cos)$	INVALID	Sem Domínio
$x \geq 7.104760e+002 (\cosh, \coshf)$	INEXACT+OVERFLOW	OVERFLOW

### LOG10()

Função utilizada para calcular o logaritmo natural de um valor numérico, em base 10.

LOG10() é uma função numérica que calcula o logaritmo natural de um número. O logaritmo natural tem como base o valor 10. Devido ao arredondamento matemático, os valores retornados por LOG() podem não coincidir exatamente.

**Sintaxe:** LOG10(nNatural)

**Parâmetros:**

nNatural	Valor cujo o logaritmo deve ser encontrado.
----------	---

**Retorno:**

Numérico	A função retorna o logaritmo de nNatural se bem sucedidas. Se nNatural for negativo, estas funções retornam um indefinido, pelo defeito. Se nNatural for 0, retornam INF(infinito).
----------	---

### SIN()

Função utilizada para calcular o valor do seno ou seno hiperbólico. Devemos informar como parâmetro para a função um valor que representa o ângulo em radianos.

**Importante:** Se  $x \geq 2^{63}$  ou  $x \leq -2^{63}$  ocorre perda significante na chamada da função SIN().

**Sintaxe:** SIN(nAngulo)

**Parâmetros:**

nAngulo	Valor do ângulo em radianos.
---------	------------------------------

**Retorno:**

Numérico	Retorna o valor do seno do ângulo especificado.
----------	---

#### Situações inválidas:

Entrada	Exceção apresentada	Significado da Exceção
$\pm \text{QNAN}, \text{IND}$	None	Sem Domínio
$\pm \infty$ (senf, sen)	INVALID	Sem Domínio
$x \geq 7.104760e+002$ (senh, senhf)	INEXACT+OVERFLOW	OVERFLOW

### SQRT()

Função utilizada para calcular a raiz quadrada de um número positivo.

**Sintaxe:** SQRT(nValor)

**Parâmetros:**

nValor	Um número positivo do qual será calculada a raiz quadrada.
--------	--

**Retorno:**

Numérico	Retorna um valor numérico calculado com precisão dupla. A quantidade de casas decimais exibidas é determinada apenas por SET DECIMALS, sem importar a configuração de SET FIXED. Um número negativo <nValor> retorna zero.
----------	--

### TAN()

Função utilizada para calcular o valor da tangente ou tangente hiperbólica.

**Importante:** Se  $x \geq 2^{63}$  ou  $x \leq -2^{63}$  ocorre perda significante na chamada da função cos.

**Sintaxe:** TAN(nAngulo)

**Parâmetros:**

nAngulo	Valor do ângulo em radianos.
---------	------------------------------

**Retorno:**

Numérico	Retorna o valor da tangente do ângulo especificado.
----------	---

#### Situações inválidas:

Entrada	Exceção apresentada	Significado da Exceção
$\pm \text{QNAN}, \text{IND}$	None	Sem Domínio
$\pm \infty$	INVALID	Sem Domínio

## Análise de variáveis

### **TYPE()**

---

Determina o tipo do conteúdo de uma variável, a qual não foi definida na função em execução.

**Sintaxe:** TYPE("cVariavel")

**Parâmetros**

<b>"cVariavel"</b>	Nome da variável que se deseja avaliar, entre aspas ("").
--------------------	---

**Exemplo:**

```
IF TYPE("dDataBase") == "D"  
    MSGINFO("Database do sistema: "+DTOC("dDataBase"))  
ELSE  
    MSGINFO("Variável indefinida no momento")  
ENDIF
```

### **VALTYPE()**

---

Determina o tipo do conteúdo de uma variável, a qual não foi definida na função em execução.

**Sintaxe:** VALTYPE(cVariavel)

**Parâmetros**

<b>cVariavel</b>	Nome da variável que se deseja avaliar.
------------------	---

**Exemplo:**

```
STATIC FUNCTION GETTEXTO(nTamanho, cTitulo, cSay)
LOCAL cTexto          := ""
LOCAL nColF, nLargGet := 0
PRIVATE oDlg
Default cTitulo := "Tela para informar texto"
Default cSay      := "Informe o texto:"
Default nTamanho    := 1

nTamanho   := IIF(ValType(nTamanho) != "N", 1, nTamanho) // Se o
parâmetro foi passado
cTexto      := Space(nTamanho); nLargGet := Round(nTamanho * 2.5, 0);
nColF       := Round(195 + (nLargGet * 1.75), 0)

DEFINE MSDIALOG oDlg TITLE cTitulo FROM 000,000 TO 120,nColF PIXEL
@ 005,005 TO 060, Round(nColF/2,0) OF oDlg PIXEL
@ 010,010 SAY cSay SIZE 55, 7 OF oDlg PIXEL
@ 010,065 MSGET cTexto SIZE nLargGet, 11 OF oDlg PIXEL ;
Picture "@!" VALID !Empty(cTexto)
DEFINE SBUTTON FROM 030, 010 TYPE 1 ;
ACTION (nOpc := 1,oDlg:End()) ENABLE OF oDlg
DEFINE SBUTTON FROM 030, 040 TYPE 2 ;
ACTION (nOpc := 0,oDlg:End()) ENABLE OF oDlg
ACTIVATE MSDIALOG oDlg CENTERED
cTexto := IIF(nOpc==1,cTexto,"")
RETURN cTexto
```

## Manipulação de arrays

### AADD()

---

A função AADD() permite a inserção de um item em um *array* já existente, sendo que este item podem ser um elemento simples, um objeto ou outro *array*.

**Sintaxe: AADD(aArray, xItem)**

**Parâmetros**

<b>aArray</b>	Array pré-existente no qual será adicionado o item definido em xItem
<b>xItem</b>	Item que será adicionado ao <i>array</i> .

**Exemplo:**

```
aDados := {} // Define que a variável aDados é um array, sem
especificar suas dimensões.
altem    := {} // Define que a variável altem    é um array, sem
especificar suas dimensões.

AADD(altem, cVariavel1) // Adiciona um elemento no array altem de
acordo com o cVariavel1
AADD(altem, cVariavel2) // Adiciona um elemento no array altem de
acordo com o cVariavel2
AADD(altem, cVariavel3) // Adiciona um elemento no array altem de
acordo com o cVariavel3

// Neste ponto o array a Item possui 03 elementos os quais podem ser
acessados com:
// altem[1] -> corresponde ao conteúdo de cVariavel1
// altem[2] -> corresponde ao conteúdo de cVariavel2
// altem[3] -> corresponde ao conteúdo de cVariavel3

AADD(aDados,altem) // Adiciona no array aDados o conteúdo do array
altem

// Neste ponto, o array a aDados possui apenas um elemento, que
também é um array
// contendo 03 elementos:
// aDados [1][1] -> corresponde ao conteúdo de cVariavel1
// aDados [1][2] -> corresponde ao conteúdo de cVariavel2
// aDados [1][3] -> corresponde ao conteúdo de cVariavel3

AADD(aDados, altem)
AADD(aDados, altem)

// Neste ponto, o array aDados possui 03 elementos, aonde cada qual
é um array com outros
// 03 elementos, sendo:

// aDados [1][1] -> corresponde ao conteúdo de cVariavel1
// aDados [1][2] -> corresponde ao conteúdo de cVariavel2
// aDados [1][3] -> corresponde ao conteúdo de cVariavel3

// aDados [2][1] -> corresponde ao conteúdo de cVariavel1
// aDados [2][2] -> corresponde ao conteúdo de cVariavel2
// aDados [2][3] -> corresponde ao conteúdo de cVariavel3

// aDados [3][1] -> corresponde ao conteúdo de cVariavel1
// aDados [3][2] -> corresponde ao conteúdo de cVariavel2
// aDados [3][3] -> corresponde ao conteúdo de cVariavel3
```

```
// Desta forma, o      array aDados montando com uma estrutura de 03  
linhas e 03 colunas, com  
// o conteúdo definido por variáveis externas, mas com a mesma forma  
obtida com o uso do  
// comando: aDados :=      ARRAY(3,3).
```

## ACLONE()

---

A função ACLONE() realiza a cópia dos elementos de um *array* para outro *array* integralmente.

**Sintaxe:** AADD(*aArray*)

### Parâmetros

***aArray***

Array pré-existente que terá seu conteúdo copiado para o *array* especificado.

### Exemplo:

```
// Utilizando o      array aDados utilizado no exemplo da função AADD()  
altens := ACLONE(aDados)  
  
// Neste ponto, o      array altens possui exatamente a mesma estrutura e  
informações do      array  
// aDados.
```



**Importante**

Por ser uma estrutura de memória, um *array* não pode ser simplesmente copiado para outro *array* através de uma atribuição simples (“:=”).

Para mais informações sobre a necessidade de utilizar o comando ACLONE() verifique o tópico 6.1.3 – Cópia de Arrays.

---

## ACOPY()

---

Função de *array* que copia elementos do *array* aOrigem para *array* aDestino. O *array* destino aDestino já deve ter sido declarado e grande o bastante para conter os elementos que serão copiados. Se o *array* aOrigem contiver mais elementos, alguns dos elementos não serão copiados. ACOPY() copia os valores de todos os dados, incluindo valores nulos (NIL) e códigos de bloco.

Se um elemento for um *subarray*, o elemento correspondente no *array* aDestino, conterá o mesmo *subarray*. Portanto, ACOPY() não produzirá uma cópia completa de *array* multidimensionais.

**Sintaxe: ACOPY( aOrigem, aDestino , [ nInicio ], [ nQtde ], [ nPosDestino ] )**

**Parâmetros:**

<b>aOrigem</b>	é o <i>array</i> que contém os elementos a serem copiados.
<b>aDestino</b>	é o <i>array</i> que receberá a cópia dos elementos.
<b>nInicio</b>	indica qual o índice do primeiro elemento de <i>aOrigem</i> que será copiado. Se não for especificado, o valor assumido será 01.
<b>nQtde</b>	indica a quantidade de elementos a serem copiados a partir do <i>array aOrigem</i> . iniciando-se a contagem a partir da posição <i>nInicio</i> . Se <i>nQtde</i> não for especificado, todos os elementos do <i>array aOrigem</i> serão copiados, iniciando-se a partir da posição <i>nInicio</i> .
<b>nPosDestino</b>	é a posição do elemento inicial no <i>array aDestino</i> que receberá os elementos de <i>aOrigem</i> . Se não especificado, será assumido 01.

**Retorno:**

<b>aDestino</b>	referência ao <i>array aDestino</i> .
-----------------	---------------------------------------

**Exemplo:**

```
LOCAL nCount := 2, nStart := 1, aOne, aTwo  
aOne := { 1, 1, 1 }  
aTwo := { 2, 2, 2 }  
ACOPY(aOne, aTwo, nStart, nCount)  
// Result: aTwo is now { 1, 1, 2 }
```

---

**ADEL()**

A função ADEL() permite a exclusão de um elemento do *array*. Ao efetuar a exclusão de um elemento, todos os demais são reorganizados de forma que a ultima posição do *array* passará a ser nula.

**Sintaxe: ADEL(aArray, nPosição)**

**Parâmetros**

<b>aArray</b>	<i>Array</i> do qual deseja-se remover uma determinada posição.
<b>nPosição</b>	Posição do <i>array</i> que será removida.

### **Exemplo:**

```
// Utilizando o      array altens do exemplo da função ACLONE() temos:  
  
ADEL(altens,1) // Será removido o primeiro elemento do      array altens.  
  
// Neste ponto, o      array altens continua com 03 elementos, aonde:  
// altens[1] -> antigo altens[2], o qual foi reordenado como efeito  
da exclusão do item 1.  
// altens[2] -> antigo altens[3], o qual foi reordenado como efeito  
da exclusão do item 1.  
// altens[3] -> conteúdo nulo, por se tratar do item excluído.
```

### **ADIR()**

Função que preenche os arrays passados com os dados dos arquivos encontrados, através da máscara informada. Tanto arquivos locais (Remote) como do servidor podem ser informados.

**Importante:** *ADir é uma função obsoleta, utilize sempre Directory().*

**Sintaxe:** ADIR([ *cArqEspec* ], [ *aNomeArq* ], [ *aTamanho* ], [ *aData* ],  
[ *aHora* ], [ *aAtributo* ])

#### **Parâmetros:**

<b>cArqEspec</b>	Caminho dos arquivos a serem incluídos na busca de informações. Segue o padrão para especificação de arquivos, aceitando arquivos no servidor Protheus e no Cliente. Caracteres como * e ? são aceitos normalmente. Caso seja omitido, serão aceitos todos os arquivos do diretório default ( *.* ).
<b>aNomeArq</b>	Array de Caracteres. É o array com os nomes dos arquivos encontrados na busca. O conteúdo anterior do array é apagado.
<b>aTamanho</b>	Array Numérico. São os tamanhos dos arquivos encontrados na busca.
<b>aData</b>	Array de Datas. São as datas de modificação dos arquivos encontrados na busca.
<b>aHora</b>	Array de Caracteres. São os horários de modificação dos arquivos encontrados. Cada elemento contém horário no formato: hh:mm:ss.
<b>aAtributos</b>	Array de Caracteres. São os atributos dos arquivos, caso esse array seja passado como parâmetros, serão incluídos os arquivos com atributos de sistema e ocultos.

#### **Retorno:**

<b>nArquivos</b>	Quantidade de arquivos encontrados.
------------------	-------------------------------------

**Exemplo:**

```
LOCAL aFiles[ADIR("*.TXT")]
ADIR("*.TXT", aFiles)
AEVAL(aFiles, { |element| QOUT(element) })
```

**AFILL()**

Função de manipulação de *arrays*, que preenche os elementos do *array* com qualquer tipo de dado. Incluindo *code-block*. Esta função não deve ser usada para preencher um *array* com outro *array*.

**Sintaxe:** AFILL( *aDestino* , *xExpValor*, [ *nInicio* ], [ *nQuantidade* ] )

**Parâmetros**

<b>aDestino</b>	É o onde os dados serão preenchidos.
<b>xExpValor</b>	É o dado que será preenchido em todas as posições informadas, não é permitida a utilização de <i>arrays</i> .
<b>nInicio</b>	É a posição inicial de onde os dados serão preenchidos, o valor padrão é 1.
<b>nCount</b>	Quantidade de elementos a partir de [ <i>nInicio</i> ] que serão preenchidos com < <i>xExpValor</i> >, caso não seja informado o valor será a quantidade de elementos até o final do <i>array</i> .

**Retorno:**

<b>aDestino</b>	Retorna uma referência para <i>aDestino</i> .
-----------------	---

**Exemplo:**

```
LOCAL aLogic[3]
// Resultado: aLogic é { NIL, NIL, NIL }
AFILL(aLogic, .F.)
// Resultado: aLogic é { .F., .F., .F. }
AFILL(aLogic, .T., 2, 2)
// Resultado: aLogic é { .F., .T., .T. }
```

## AINS()

---

A função AINS() permite a inserção de um elemento no array especificado em qualquer ponto da estrutura do mesmo, diferindo desta forma da função AADD() a qual sempre insere um novo elemento ao final da estrutura já existente.

**Sintaxe: AINS(aArray, nPosicao)**

**Parâmetros**

<b>aArray</b>	Array pré-existente no qual desejasse inserir um novo elemento.
<b>nPosicao</b>	Posição na qual o novo elemento será inserido.

**Exemplo:**

```
aAlunos := {"Edson", "Robson", "Renato", "Tatiana"}  
  
AINS(aAlunos,3)  
// Neste ponto o array aAlunos terá o seguinte conteúdo:  
// {"Edson", "Robson", nulo, "Renato", "Tatiana"}
```



*Importante*

Similar ao efeito da função ADEL(), o elemento inserido no *array* pela função AINS() terá um conteúdo nulo, sendo necessário trata-lo após a realização deste comando.

---

## ARRAY()

---

A função Array() é utilizada na definição de variáveis de tipo *array*, como uma opção a sintaxe utilizando chaves ("{}").

**Sintaxe: Array(nLinhas, nColunas)**

**Parâmetros**

<b>nLinhas</b>	Determina o número de linhas com as quais o <i>array</i> será criado.
<b>nColunas</b>	Determina o número de colunas com as quais o <i>array</i> será criado.

**Exemplo:**

```
aDados := Array(3,3) // Cria um array de três linhas, cada qual com  
3 colunas.
```



*Importante*

O *array* definido pelo comando *Array()* apesar de já possuir a estrutura solicitada, não possui conteúdo em nenhum de seus elementos, ou seja:

aDados[1] -> *array* de três posições

aDados[1][1] -> posição válida, mas de conteúdo nulo.

## ASCAN()

A função ASCAN() permite que seja identificada a posição do *array* que contém uma determinada informação, através da análise de uma expressão descrita em um bloco de código.

**Sintaxe: ASCAN(aArray, bSeek)**

**Parâmetros**

<b>aArray</b>	<i>Array</i> pré-existente no qual desejasse identificar a posição que contém a informação pesquisada.
<b>bSeek</b>	Bloco de código que configura os parâmetros da busca a ser realizada.

**Exemplo:**

```
aAlunos := {"Márcio", "Denis", "Arnaldo", "Patrícia"}  
  
bSeek := {|x| x == "Denis"}  
  
nPosAluno := aScan(aAlunos,bSeek) // retorno esperado
```

¶ 2



*Dica*

Durante a execução da função aScan, a variável “x” receberá o conteúdo o item que está posicionado no momento, no caso aAlunos[x]. Como aAlunos[x] é uma posição do *array* que contém o nome do aluno, “x” poderia ser renomeada para cNome, e a definição do bloco bSeek poderia ser escrita como:

bSeek := {|cNome| cNome == "Denis"}

Na definição dos programas é sempre recomendável utilizar variáveis com nomes significativos, desta forma os blocos de código não são exceção.



*Dica*

Sempre opte por analisar como o bloco de código será utilizado e ao invés de “x”, “y” e similares, defina os parâmetros com nomes que representem seu conteúdo. Será mais simples o seu entendimento e o entendimento de outros que forem analisar o código escrito.

## ASCANX()

---

Função utilizada para varrer um vetor procurando um valor especificado, operando de forma similar a função ASCAN.

A diferença fundamental da função ASCANX é que esta função recebe um segundo parâmetro em seu *code-block* representando o índice do *array*.

**Sintaxe:** ASCANX ( < xDestino > , < bSeek > , [ nInicio ] , [ nCont ] )

**Parâmetros:**

<b>xDestino</b>	Representa o objeto a ser varrido pela função, pode ser atribuído ao parâmetro um <i>array</i> um Objeto.
<b>bSeek</b>	Representa o valor que será pesquisado, podendo ser um bloco de código.
<b>nInicio</b>	Representa o elemento a partir do qual terá inicio a pesquisa, quando este argumento não for informado o valor <i>default</i> será 1.
<b>nCont</b>	Representa a quantidade de elementos que serão pesquisados a partir da posição inicial, quando este argumento não for informado todos elementos do <i>array</i> serão pesquisados.

**Exemplo.:**

```
nPos := aScanX( ARRAY, { |X,Y| X[1] == cNome .OR. y<=100})
```

---

No código demonstrado acima, note a inclusão no *code-block* do *Y*, onde a função irá terminar sua execução em 3 condições:



*Dica*

1) Até encontrar o elemento no *ARRAY* com a ocorrência *cNome*, retornando a posição desse elemento.

2) Essa é novidade, ASCANX irá verificar o *Array* até a posição 100.

3) O elemento *cNome* não foi encontrado no *ARRAY* e a condição de *Y* até 100 não satisfaz, pois o *array* é menor do que 100 posições!



*Dica*

---

Como ASCAN() que utiliza o operador (=) para comparações, a função ASCANX() também é case sensitive, no caso os elementos procurados devem ser exatamente igual.

## ASIZE()

---

A função ASIZE permite a redefinição da estrutura de um array pré-existente, adicionando ou removendo itens do mesmo.

### Sintaxe: ASIZE(aArray, nTamanho)

#### Parâmetros

<b>aArray</b>	Array pré-existente que terá sua estrutura redimensionada.
<b>nTamanho</b>	Tamanho com o qual deseja-se redefinir o array. Se o tamanho for menor do que o atual, serão removidos os elementos do final do array, já se o tamanho for maior do que o atual serão inseridos itens nulos ao final do array.

#### Exemplo:

```
// Utilizando o array altens, o qual teve um elemento excluído pelo uso da função ADEL()  
  
ASIZE(altens,Len(altens-1))  
  
// Neste ponto o array altens possui 02 elementos, ambos com conteúdos válidos.
```



Dica

Utilizar a função ASIZE() após o uso da função ADEL() é uma prática recomendada e evita que seja acessada uma posição do array com um conteúdo inválido para a aplicação em uso.

---

## ASORT()

---

A função ASORT() permite que os itens de um array sejam ordenados a partir de um critério pré-estabelecido.

### Sintaxe: ASORT(aArray, nInicio, nItens, bOrdem)

#### Parâmetros

<b>aArray</b>	Array pré-existente que terá seu conteúdo ordenado através de um critério estabelecido.
<b>nInicio</b>	Posição inicial do array para início da ordenação. Caso não seja informado, o array será ordenado a partir de seu primeiro elemento.
<b>nItens</b>	Quantos itens, a partir da posição inicial deverão ser ordenados. Caso não seja informado, serão ordenados todos os elementos do array.
<b>bOrdem</b>	Bloco de código que permite a definição do critério de ordenação do array. Caso bOrdem não seja informado, será utilizado o critério ascendente.

---

Um bloco de código é basicamente uma função escrita em linha. Desta forma sua estrutura deve “suportar” todos os requisitos de uma função, os quais são através da análise e interpretação de parâmetros recebidos, executar um processamento e fornecer um retorno.

Com base nesse requisito, pode-se definir um bloco de código com a estrutura abaixo:



Dica

bBloco := { |xPar1, xPar2, ... xParZ| Ação1, Ação2, AçãoZ }, aonde:

| | -> define o intervalo onde estão compreendidos os parâmetros

Ação Z-> expressão que será executadas pelo bloco de código

Ação1... AçãoZ -> intervalo de expressões que serão executadas pelo bloco de código, no formato de lista de expressões.

Retorno -> resultado da ultima ação executada pelo bloco de código, no caso AçãoZ.

Para maiores detalhes sobre a estrutura e utilização de blocos de código consulte o tópico 6.2 – Listas de Expressões e Blocos de código.

---

### Exemplo 01 – Ordenação ascendente

```
aAlunos := { "Mauren", "Soraia", "Andréia"}  
aSort(aAlunos)  
// Neste ponto, os elementos do array aAlunos serão {"Andréia",  
"Mauren", "Soraia"}
```

## **Exemplo 02 – Ordenação descendente**

```
aAlunos := { "Mauren", "Soraia", "Andréia"}  
bOrdem := {|x,y| x > y }  
  
// Durante a execução da função aSort(), a variável "x" receberá o  
conteúdo do item que está  
// posicionado. Como o item que está posicionado é a posição  
aAlunos[x] e aAlunos[x] ->  
// string contendo o nome de um aluno, pode-se substituir "x" por  
cNomeAtu.  
// A variável "y" receberá o conteúdo do próximo item a ser  
avaliado, e usando a mesma  
// analogia de "x", pode-se substituir "y" por cNomeProx. Desta  
forma o bloco de código  
// bOrdem pode ser re-escrito como:  
  
bOrdem := {|cNomeAtu, cNomeProx| cNomeAtu > cNomeProx}  
  
aSort(aAlunos,,bOrdem)  
  
// Neste ponto, os elementos do array aAlunos serão {"Soraia",  
"Mauren", "Andréia"}
```

## **ATAIL()**

ATAIL() é uma função de manipulação de *array* que retorna o último elemento de um *array*. Ela deve ser usada em substituição da seguinte construção: aArray [LEN( aArray )]

**Sintaxe:** ATAIL( *aArray* )

**Parâmetros:**

<b>aArray</b>	É o <i>array</i> de onde será retornado o último elemento.
---------------	--

**Retorno:**

<b>nUltimo</b>	Número do último elemento do <i>array</i> .
----------------	---

**Exemplo:**

```
aArray := {"a", "b", "c", "d"}  
ATAIL(aArray) // Resultado: d
```

## **Manipulação de blocos de código**

## EVAL()

---

A função EVAL() é utilizada para avaliação direta de um bloco de código, utilizando as informações disponíveis no mesmo de sua execução. Esta função permite a definição e passagem de diversos parâmetros que serão considerados na interpretação do bloco de código.

**Sintaxe:** EVAL(bBloco, xParam1, xParam2, xParamZ)

### Parâmetros

<b>bBloco</b>	Bloco de código que será interpretado.
<b>xParamZ</b>	Parâmetros que serão passados ao bloco de código. A partir da passagem do bloco, todos os demais parâmetros da função serão convertidos em parâmetros para a interpretação do código.

### Exemplo:

```
nInt := 10
bBloco := {|N| x:= 10, y:= x*N, z:= y/(x*N)}
nValor := EVAL(bBloco, nInt)
// O retorno será dado pela avaliação da ultima ação da lista de
expressões, no caso "z".
// Cada uma das variáveis definidas em uma das ações da lista de
expressões fica disponível
// para a próxima ação.
// Desta forma temos:
// N  ↗ recebe nInt como parâmetro (10)
// X  ↗ tem atribuído o valor 10 (10)
// Y  ↗ resultado da multiplicação de X por N (100)
// Z  ↗ resultado a divisão de Y pela multiplicação de X por N ( 100
/ 100)  ↗ 1
```

## DBEVAL()

---

A função DBEval() permite que todos os registro de uma determinada tabela sejam analisados e para cada registro será executado o bloco de código definido.

**Sintaxe:** Array(bBloco, bFor, bWhile)

### Parâmetros

<b>bBloco</b>	Bloco de código principal, contendo as expressões que serão avaliadas para cada registro do alias ativo.
<b>bFor</b>	Condição para continuação da análise dos registros, com o efeito de uma

	estrutura <i>For ... Next</i> .
<b>bWhile</b>	Condição para continuação da análise dos registros, com o efeito de uma estrutura <i>While ... End</i> .

### Exemplo 01:

```
// Considerando o trecho de código abaixo:  
dbSelectArea("SX5")  
dbSetOrder(1)  
dbGotop()  
  
While !Eof() .And. X5_FILIAL == xFilial("SX5") .And.; X5_TABELA <= mv_par02  
    nCnt++  
    dbSkip()  
End  
  
// O mesmo pode ser re-escrito com o uso da função DBEVAL():  
dbEval( { |x| nCnt++ }, { |X5_FILIAL==xFilial("SX5") .And.  
X5_TABELA<=mv_par02} )
```

### Exemplo 02:

```
// Considerando o trecho de código abaixo:  
dbSelectArea("SX5")  
dbSetOrder(1)  
dbGotop()  
  
While !Eof() .And. X5_TABELA == cTabela  
    AADD(aTabela,{X5_CHAVE, Capital(X5_DESCRI)})  
    dbSkip()  
End  
  
// O mesmo pode ser re-escrito com o uso da função DBEVAL():  
  
dbEval({ | | AADD(aTabela,{X5_CHAVE,Capital(X5_DESCRI)})}, { | |  
X5_TABELA==cTabela})
```



*Importante*

Na utilização da função DBEVAL() deve ser informado apenas um dos dois parâmetros: bFor ou bWhile.

## **AEVAL()**

A função AEVAL() permite que todos os elementos de um determinada *array* sejam analisados e para cada elemento será executado o bloco de código definido.

**Sintaxe: AEVAL(aArray, bBloco, nInicio, nFim)**

### **Parâmetros**

<b>aArray</b>	Array que será avaliado na execução da função.
<b>bBloco</b>	Bloco de código principal, contendo as expressões que serão avaliadas para cada elemento do <i>array</i> informado.
<b>nInicio</b>	Elemento inicial do <i>array</i> , a partir do qual serão avaliados os blocos de código.
<b>nFim</b>	Elemento final do <i>array</i> , até o qual serão avaliados os blocos de código.

### **Exemplo 01:**

Considerando o trecho de código abaixo:

```
AADD(aCampos,"A1_FILIAL")
AADD(aCampos,"A1_COD")
SX3->(dbSetOrder(2))
For nX:=1 To Len(aCampos)
    SX3->(dbSeek(aCampos[nX]))
    AADD(aTitulos,AllTrim(SX3->X3_TITULO))
Next nX
```

O mesmo pode ser re-escrito com o uso da função AEVAL():

```
aEval(aCampos,{|x| SX3->(dbSeek(x)),IIF(Found(), AADD(aTitulos,;
AllTrim(SX3->X3_TITULO))))}
```

## **Manipulação de strings**

### **ALLTRIM()**

Retorna uma *string* sem os espaços à direita e à esquerda, referente ao conteúdo informado como parâmetro.

A função ALLTRIM() implementa as ações das funções RTRIM (“right trim”) e LTRIM (“left trim”).

**Sintaxe: ALLTRIM(cString)**

## Parâmetros

<b>cString</b>	<i>String que será avaliada para remoção dos espaços a direita e a esquerda.</i>
----------------	--

### Exemplo:

```
cNome := ALLTRIM(SA1->A1_NOME)

MSGINFO("Dados do campo A1_NOME:" +CRLF
" Tamanho:" + CVALTOCHAR(LEN(SA1->A1_NOME))+CRLF
"Texto." + CVALTOCHAR(LEN(cNome)))
```

## ASC()

Converte uma informação caractere em seu valor de acordo com a tabela ASCII.

### Sintaxe: ASC(cCaractere)

## Parâmetros

<b>cCaractere</b>	<i>Caracter que será consultado na tabela ASCII.</i>
-------------------	--

### Exemplo:

```
USER FUNCTION NoAcento(Arg1)
Local nConta := 0
Local cLetra := ""
Local cRet    := ""
Arg1 := Upper(Arg1)
For nConta:= 1 To Len(Arg1)
    cLetra := SubStr(Arg1, nConta, 1)
    Do Case
        Case (Asc(cLetra) > 191 .and. Asc(cLetra) < 198) .or. ;
            (Asc(cLetra) > 223 .and. Asc(cLetra) < 230)
                cLetra := "A"
        Case (Asc(cLetra) > 199 .and. Asc(cLetra) < 204) .or. ;
            (Asc(cLetra) > 231 .and. Asc(cLetra) < 236)
                cLetra := "E"
        Case (Asc(cLetra) > 204 .and. Asc(cLetra) < 207) .or. ;
            (Asc(cLetra) > 235 .and. Asc(cLetra) < 240)
                cLetra := "I"
        Case (Asc(cLetra) > 209 .and. Asc(cLetra) < 215) .or. ;
            (Asc(cLetra) == 240) .or. (Asc(cLetra) > 241 .and. Asc(cLetra) <
247)
```

```

cLetra := "O"

Case (Asc(cLetra) > 216 .and. Asc(cLetra) < 221) .or.;
(Asc(cLetra) > 248 .and. Asc(cLetra) < 253)
    cLetra := "U"

Case Asc(cLetra) == 199 .or. Asc(cLetra) == 231
    cLetra := "C"

EndCase

cRet := cRet+cLetra

Next

Return UPPER(cRet)

```

## AT()

---

Retorna a primeira posição de um caracter ou *string* dentro de outra *string* especificada.

**Sintaxe:** AT(*cCaractere, cString* )

### Parâmetros

<b>cCaractere</b>	Caractere ou <i>string</i> que se deseja verificar.
<b>cString</b>	<i>String</i> na qual será verificada a existência do conteúdo de <i>cCaractere</i> .

### Exemplo:

```

STATIC FUNCTION NOMASCARA(cString,cMascara,nTamanho)

LOCAL cNoMascara          := ""
LOCAL nX := 0

IF !Empty(cMascara) .AND. AT(cMascara,cString) > 0
    FOR nX := 1 TO Len(cString)
        IF !(SUBSTR(cString,nX,1) $ cMascara)
            cNoMascara += SUBSTR(cString,nX,1)
        ENDIF
        NEXT nX
        cNoMascara := PADR(ALLTRIM(cNoMascara),nTamanho)
    ELSE
        cNoMascara := PADR(ALLTRIM(cString),nTamanho)
    ENDIF

RETURN cNoMascara

```

## BITON()

Função utilizada para ligar determinados bits de uma *String* passada por parâmetro para a função. Além da *string* a ser alterada, a função também recebe como parâmetro um numérico que indica o *bit* de início a ser alterado, um numérico que indica a quantidade de *bits* a serem alterados(ligados) e o tamanho da *string* passada.

**Sintaxe:** BITON ( < cValue > , < nBitIni > , < nBitEnd > , < nStrLen > )

### Parâmetros

<b>cValue</b>	<i>String</i> no qual desejamos ligar os <i>bits</i> .
<b>nBitIni</b>	Indica a partir de qual <i>bit</i> , começará a ser ligados os <i>bits</i> na <i>String</i> .
<b>nBitEnd</b>	Indica a quantidade de <i>bits</i> que serão ligados a partir do início.
<b>nStrLen</b>	Representa o tamanho da <i>String</i> passada para a função.

## CAPITAL()

Função que avalia a *string* passada como parâmetro alterando a primeira letra de cada palavra para maiúscula e as demais letras como minúsculas.

**Sintaxe:** CAPITAL(cFrase)

### Parâmetros:

<b>cFrase</b>	<i>String</i> a ser avaliada.
---------------	-------------------------------

### Retorno:

<b>String</b>	Conteúdo da <i>string</i> original com as modificações necessárias para atender a condição da função.
---------------	---

## CHR()

Converte um valor número referente a uma informação da tabela ASCII no caractere que esta informação representa.

**Sintaxe:** CHR(nASCII)

### Parâmetros

<b>nASCII</b>	Código ASCII do caractere
---------------	---------------------------

**Exemplo:**

```
#DEFINE CRLF CHR(13)+CHR(10) // FINAL DE LINHA
```

**DESCEND()**

---

Função de conversão que retorna a forma complementada da expressão *string* especificada. Esta função normalmente é utilizada para a criação de indexadores em ordem decrescente

**Sintaxe:** DESCEND ( < cString > )

**Parâmetros:**

cString	Corresponde à seqüência de caracteres a ser analisada.
---------	--

**Retorno:**

Caracter	<i>String</i> complementada da <i>string</i> analisada.
----------	---

**Exemplo:**

```
// Este exemplo utiliza DESCEND() em uma expressão INDEX para criar
um índice de datas de
// ordem descendente:

USE Sales NEW
INDEX ON DESCEND(DTOS(OrdDate)) TO SalesDate

// Depois, DESCEND() pode ser utilizado para fazer uma pesquisa
(SEEK) no índice
// descendente:

DbSEEK(DESCEND(DTOS(dFindDate)))
```

**GETDTOVAL()**

---

Função utilizada para retornar um numero formatado, de acordo com o valor passado por parâmetro, sendo que irá apenas manter os valores numéricos contidos na *string* passada por parâmetro, verificando se existe algum caractere '.' retornando um número fracionário, na ordem dos números contidos na *string*.

A função é muito útil quando desejamos utilizar o valor numérico de uma data que está contida em uma *string*.

## Sintaxe: GETDTOVAL ( < cDtoVal > )

### Parâmetros:

cDtoVal	Representa uma <i>string</i> contendo um valor numérico no qual será convertido.
---------	--

### Retorno:

Numérico	Retorna um dado numérico de acordo com o valor informado em <cDtoVal>.
----------	--

### Exemplo:

```
GetDtoVal('123456')      //retorno 123456.0000
GetDtoVal('1/2/3/4/5/6')  //retorno 123456.0000
GetDtoVal('fim.123456')   //retorno 0.123456
GetDtoVal('teste')        //retorno 0.0
```

## ISALPHA()

Função utilizada para determinar se o caractere mais à esquerda em uma cadeia de caracteres é alfabético, permitindo avaliar se o *string* especificado começa com um caractere alfabético. Um caractere alfabético consiste em qualquer letra maiúscula ou minúscula de “A” a “Z”.

## Sintaxe: ISALPHA ( < cString > )

### Parâmetros:

cString	Cadeia de caracteres a ser examinada.
---------	---------------------------------------

### Retorno:

Lógico	Retorna verdadeiro (.T.) se o primeiro caractere em <cString> for alfabético, caso contrário, retorna falso (.F.).
--------	--

## ISDIGIT()

Função utilizada para determinar se o caractere mais à esquerda em uma cadeia de caracteres é um dígito, permitindo avaliar se o primeiro caractere em um *string* é um dígito numérico entre zero e nove.

## Sintaxe: ISDIGIT ( < cString > )

**Parâmetros:**

<b>cString</b>	Cadeia de caracteres a ser examinada.
----------------	---------------------------------------

**Retorno:**

<b>Lógico</b>	Retorna verdadeiro (.T.) caso o primeiro caractere da cadeia seja um dígito entre zero e nove; caso contrário, retorna falso (.F.).
---------------	---

## **ISLOWER()**

---

Função utilizada para determinar se o caractere mais à esquerda é uma letra minúscula, permitindo avaliar se o primeiro caractere de um *string* é uma letra minúscula. É o contrário de ISUPPER(), a qual determina se a cadeia de caracteres começa com uma letra maiúscula. ISLOWER() e ISUPPER() ambas são relacionadas às funções LOWER() e UPPER(), que convertem caracteres minúsculos para maiúsculos, e vice-versa.

**Sintaxe:** ISLOWER( < cString > )

**Parâmetros:**

<b>cString</b>	Cadeia de caracteres a ser examinada.
----------------	---------------------------------------

**Retorno:**

<b>Lógico</b>	Retorna verdadeiro (.T.) caso o primeiro caractere da cadeia seja minúsculo, caso contrário, retorna falso (.F.).
---------------	---

## **ISUPPER()**

---

Função utilizada para determinar se o caractere mais à esquerda é uma letra maiúscula, permitindo avaliar se o primeiro caractere de um *string* é uma letra maiúscula. É o contrário de ISLOWER(), a qual determina se a cadeia de caracteres começa com uma letra minúscula. ISLOWER() e ISUPPER() ambas são relacionadas às funções LOWER() e UPPER(), que convertem caracteres minúsculos para maiúsculos, e vice-versa.

**Sintaxe:** ISUPPER( < cString > )

**Parâmetros:**

<b>cString</b>	Cadeia de caracteres a ser examinada.
----------------	---------------------------------------

**Retorno:**

<b>Lógico</b>	Retorna verdadeiro (.T.) caso o primeiro caractere da cadeia seja maiúsculo, caso contrário, retorna falso (.F.).
---------------	---

## **LEN()**

---

Retorna o tamanho da *string* especificada no parâmetro.

.

### **Sintaxe: LEN(cString)**

.

#### **Parâmetros**

<b>cString</b>	<i>String</i> que será avaliada.
----------------	----------------------------------

#### **Exemplo:**

```
cNome := ALLTRIM(SA1->A1_NOME)
MSGINFO("Dados do campo A1_NOME:" +CRLF
" Tamanho:" + CVALTOCHAR(LEN(SA1->A1_NOME))+CRLF
"Texto:" + CVALTOCHAR(LEN(cNome)))
```

## **LOWER()**

---

Retorna uma *string* com todos os caracteres minúsculos, tendo como base a *string* passada como parâmetro.

.

### **Sintaxe: LOWER(cString)**

.

#### **Parâmetros**

<b>cString</b>	<i>String</i> que será convertida para caracteres minúsculos.
----------------	---

#### **Exemplo:**

```
cTexto := "ADVPL"
MSGINFO("Texto:" +LOWER(cTexto))
```

## **LTRIM()**

---

Função para tratamento de caracteres utilizada para formatar cadeias de caracteres que possuam espaços em branco à esquerda. Pode ser o caso de, por exemplo, números convertidos para cadeias de caracteres através da função STR().

LTRIM() é relacionada a RTRIM(), a qual remove espaços em branco à direita, e a ALLTRIM(), que remove espaços tanto à esquerda quanto à direita.

O contrário de ALLTRIM(), LTRIM(), e RTRIM() são as funções PADC(), PADR(), e PADL(), as quais centralizam, alinham à direita, ou alinham à esquerda as cadeias de caracteres, através da inserção de caracteres de preenchimento.

**Sintaxe: LTRIM ( < cString > )**

**Parâmetros:**

<b>cString</b>	<cString> é a cadeia de caracteres a ser copiada sem os espaços em branco à esquerda.
----------------	---

**Retorno:**

<b>Caracter</b>	LTRIM() retorna uma cópia de <cString>, sendo que os espaços em branco à esquerda foram removidos. Caso <cString> seja uma cadeia de caracteres nula ("") ou toda composta de espaços em branco, LTRIM() retorna uma cadeia de caracteres nula ("").
-----------------	--

## MATHC()

---

Função utilizada para realizar operações matemáticas com *strings* que contém um valor numérico. MATHC() realiza algumas operações matemáticas como: Soma, Subtração, Divisão, Multiplicação e Exponenciação.

A função irá retornar uma *string* contendo o resultado da operação matemática, com uma especificação de até 18 casas de precisão no número.

**Sintaxe: MATHC ( < cNum1 > , < cOperacao > , < cNum2 > )**

**Parâmetros:**

<b>cNum1</b>	<i>String</i> contendo um valor numérico, representando o numero no qual desejamos realizar uma operação.
<b>cOperacao</b>	Representa a <i>string</i> que indica a operação que desejamos realizar. Olhar na tabela para verificar quais valores devem ser informados aqui.
<b>cNum2</b>	<i>String</i> contendo um valor numérico, representando o numero no qual desejamos realizar uma operação.

**Retorno:**

<b>Caracter</b>	Retorna uma nova <i>string</i> contendo o resultado matemático da operação.
-----------------	---

## OEMTOANSI()

---

Função que transforma uma *string* no Formato OEM / MS-DOS Text para uma *string ANSI Text* ( formato do Windows ).

Quando utilizamos um programa baseado no MS-DOS para alimentar uma base de dados, os acentos e caracteres especiais são gravados como texto OEM. Para tornar possível a correta visualização destes dados em uma interface Windows, utilizamos a função OemToAnsi() para realizar a conversão.

Ao utilizarmos um programa baseado no Windows para alimentar uma base de dados, o texto é capturado no formato ANSI Text. Caso este texto seja utilizado para alimentar uma base de dados a ser acessada através de um programa MS-DOS, devemos converter o dado para OEM antes de gravá-lo, através da função AnsiToOem().

.

**Sintaxe:** `OemToAnsi (<cStringOEM>)`

.

**Parâmetros:**

<b>cStringOEM</b>	<i>String</i> em formato OEM - MsDos a ser convertida.
-------------------	--

.

**Retorno:**

<b>Caracter</b>	<i>String</i> convertida para ser exibida no Windows ( Formato ANSI ).
-----------------	--

## PADL() / PADR() / PADC()

---

Funções de tratamento de *strings* que inserem caracteres de preenchimento para completar um tamanho previamente especificado em vários formatos como data ou numéricos.

- PADC() centraliza <cExp>, adicionando caracteres de preenchimento à direita e à esquerda.
- PADL() adiciona caracteres de preenchimento à esquerda.
- PADR() adiciona caracteres de preenchimento à direita.

Caso o tamanho de <cExp> exceda o argumento <nTamanho>, todas as funções PAD() truncam *string* preenchida ao <nTamanho> especificado.

PADC(), PADL(), e PADR() são utilizadas para exibir cadeias de caracteres de tamanho variável em uma área de tamanho fixo. Elas podem ser usadas, por exemplo, para assegurar o alinhamento com comandos consecutivos. Outra utilização é exibir textos em uma tela de tamanho fixo, para certificar-se de que o texto anterior foi completamente sobrescreto.

PADC(), PADL(), e PADR() são o contrário das funções ALLTRIM(), LTRIM(), e RTRIM(), as quais eliminam espaços em branco à esquerda e à direita de cadeias de caracteres.

**Sintaxe: PADL / PADR / PADC ( < cExp > , < nTamanho > , [ cCaracPreench ] )**

#### Parâmetros

<b>cExp</b>	Caractere, data, ou numérico no qual serão inseridos caracteres de preenchimento.
<b>nTamanho</b>	Tamanho da cadeia de caracteres a ser retornada.
<b>cCaracPreench</b>	Caractere a ser inserido em cExp. Caso não seja especificado, o padrão é o espaço em branco.

#### Retorno:

<b>Caracter</b>	Retornam o resultado de <cExp> na forma de uma cadeia de caracteres preenchida com <cCaracPreench>, para totalizar o tamanho especificado por <nTamanho>.
-----------------	---

## RAT()

---

Retorna a última posição de um caractere ou *string* dentro de outra *string* especificada.

**Sintaxe: RAT(cCaractere, cString)**

#### Parâmetros

<b>cCaractere</b>	Caractere ou <i>string</i> que se deseja verificar
<b>cString</b>	<i>String</i> na qual será verificada a existência do conteúdo de cCaractere.

## REPLICATE()

---

A função Replicate() é utilizada para gerar uma cadeira de caracteres repetidos a partir de um caractere base informado, podendo a *string* gerada conter até 64KB. Caso seja especificado no parâmetro de itens a repetir o número zero, será retornada uma *string* vazia.

**Sintaxe: REPLICATE(cString, nCount)**

#### Parâmetros:

<b>cString</b>	Caracter que será repetido.
<b>nCount</b>	Quantidade de ocorrências do caractere base que serão geradas na <i>string</i> de destino.

**Retorno:**

<b>cReplicated</b>	<i>String</i> contendo as ocorrências de repetição geradas para o caracter informado.
--------------------	---

---

## RTRIM()

---

Função para tratamento de caracteres utilizada para formatar cadeias de caracteres que contenham espaços em branco à direita. Ela é útil quando você deseja eliminar espaços em branco à direita ao se concatenar cadeias de caracteres. É o caso típico de campos de banco de dados que são armazenados em formato de tamanho fixo. Por exemplo, você pode usar RTRIM() para concatenar o primeiro e o último campos de nome para formar uma cadeia de caracteres de nome.

LTRIM() é relacionada a RTRIM(), que remove espaços em branco à direita, e a ALLTRIM(), que remove espaços em branco à direita e à esquerda.

O contrário de ALLTRIM(), LTRIM(), e RTRIM() são as funções PADC(), PADR(), e PADL(), as quais centralizam, alinham à direita, ou alinham à esquerda cadeias de caracteres, inserindo caracteres de preenchimento.

**Sintaxe:** RTRIM ( < cString > ) --> cTrimString

**Parâmetros:**

<b>cString</b>	<cString> é a cadeia de caracteres a ser copiada sem os espaços em branco à direita.
----------------	--

**Retorno:**

<b>Caracter</b>	RTRIM() retorna uma cópia de <cString>, sendo que os espaços em branco à direita foram removidos. Caso <cString> seja uma cadeia de caracteres nula ("") ou totalmente composta por espaços, RTRIM() retorna uma cadeia de caracteres nula ("").
-----------------	--

---

## SPACE()

---

Função de tratamento de caracteres utilizada para retornar uma quantidade especificada de espaços. A utilização desta função tem o mesmo efeito que REPLICATE(' ', <nCont>), e é normalmente utilizada para inicializar uma variável do tipo caractere, antes que a mesma seja associada a um GET.

**Sintaxe:** SPACE ( <nCont> )

**Parâmetros:**

<b>nCont</b>	A quantidade de espaços a serem retornados, sendo que o número máximo é 65.535 (64K).
--------------	---

**Retorno:**

<b>Caracter</b>	Retorna uma cadeia de caracteres. Se <nCont> for zero, SPACE() retorna uma cadeia de caracteres nula ("").
-----------------	--

## **STRTOKARR()**

---

Função utilizada para retornar um *array*, de acordo com os dados passados como parâmetro para a função. Esta função recebe uma *string* <cValue> e um caracter <cToken> que representa um separador, e para toda ocorrência deste separador em <cValue> é adicionado um item no *array*.

**Sintaxe:** STRTOKARR ( <cValue> , <cToken> )

**Parâmetros:**

<b>cValue</b>	Representa a cadeia de caracteres no qual desejamos separar de acordo com <cToken>.
<b>cToken</b>	Representa o caracter que indica o separador em <cValue>.

**Retorno:**

<b>Array</b>	<i>Array</i> de caracteres que representa a <i>string</i> passada como parâmetro.
--------------	---

**Exemplo:**

```
STRTOKARR('1;2;3;4;5', ';') //retorna {'1','2','3','4','5'}
```

## **STRTRAN()**

---

Função utilizada para realizar a busca da ocorrência da *string*, sendo *case sensitive*.

**Sintaxe:** STRTRAN ( <cString> , <cSearch> , [ cReplace ] , [ nStart ] , [ nCount ] )

**Parâmetros:**

<b>cString</b>	Seqüência de caracteres ou campo memo a ser pesquisado.
<b>cSearch</b>	Seqüência de caracteres a ser procurada em cString.
<b>cReplace</b>	Seqüência de caracteres que deve substituir a <i>string</i> cSearch. Caso não seja especificado, as ocorrências de cSearch em cString serão substituídas por uma <i>string</i> nula ("").
<b>nStart</b>	nStart corresponde ao número seqüencial da primeira ocorrência de cSearch em cString a ser substituída por cReplace. Se este argumento for omitido, o default é 1 ( um ). Caso seja passado um número menor que 1, a função retornará uma <i>string</i> em branco ("").
<b>nCount</b>	nCount corresponde ao número máximo de trocas que deverá ser realizada pela função. Caso este argumento não seja especificado, o <i>default</i> é substituir todas as ocorrências encontradas.

**Retorno:**

<b>Code-Block</b>	A função STRTRAN retorna uma nova <i>string</i> , com as ocorrências especificadas de cSearch trocadas para cReplace, conforme parametrização.
-------------------	--

**STUFF()**

---

Função que permite substituir um conteúdo caractere em uma *string* já existente, especificando a posição inicial para esta adição e o número de caracteres que serão substituídos.

**Sintaxe: STUFF(cString, nPosInicial, nExcluir, cAdicao)**

**Parâmetros:**

<b>cString</b>	A cadeia de caracteres destino na qual serão eliminados e inseridos caracteres.
<b>nPosInicial</b>	A posição inicial na cadeia de caracteres destino onde ocorre a inserção/eliminação.
<b>nExcluir</b>	A quantidade de caracteres a serem eliminados.
<b>cAdicao</b>	A cadeia de caracteres a ser inserida.

**Retorno:**

<b>Caracter</b>	Retorna a nova <i>string</i> gerada pela função com as modificações.
-----------------	--

**Exemplo:**

```
cLin := Space(100)+cEOL // Cria a          string base
cCpo := PADR(SA1->A1_FILIAL,02) // Informação que será armazenada na
string
cLin := Stuff(cLin,01,02,cCpo) // Substitui o conteúdo de cCpo na
string      base
```

**SUBSTR()**

---

Retorna parte do conteúdo de uma *string* especificada, de acordo com a posição inicial deste conteúdo na *string* e a quantidade de caracteres que deverá ser retornada a partir daquele ponto (inclusive).

**Sintaxe: SUBSTR(cString, nPosInicial, nCaracteres)**

**Parâmetros**

<b>cString</b>	<i>String</i> que se deseja verificar
<b>nPosInicial</b>	Posição inicial da informação que será extraída da <i>string</i>
<b>nCaracteres</b>	Quantidade de caracteres que deverá ser retornada a partir daquele ponto (inclusive).

**Exemplo:**

```
cCampo := "A1_NOME"
nPosUnder := AT(cCampo)
cPrefixo := SUBSTR(cCampo,1, nPosUnder) //           "A1_"
```

**TRANSFORM()**

---

Função de conversão que formata valores caractere, data, lógicos e numéricos conforme um *string* de máscara especificado, a qual inclui uma combinação de *strings* de template e funções de *picture*. Ela faz o mesmo que a cláusula PICTURE do comando @...SAY, sendo normalmente utilizada para formatar dados a serem enviados à tela ou à impressora.

**Sintaxe: TRANSFORM ( < cExp > , < cSayPicture > )**

#### Parâmetros:

<b>cExp</b>	O valor a ser formatado. Esta expressão pode ser qualquer tipo de dados válidos, exceto vetor, bloco de código, e NIL.
<b>cSayPicture</b>	Uma string de caracteres de máscara e template usado para descrever o formato da cadeia de caracteres a ser retornada.

#### Retorno:

-	Retorna a conversão de <cExp> para uma cadeia de caracteres formatada conforme a definição em <cSayPicture>.
---	--

## UPPER()

---

Retorna uma *string* com todos os caracteres maiúsculos, tendo como base a *string* passada como parâmetro.

#### Sintaxe: UPPER(cString)

#### Parâmetros

<b>cString</b>	<i>String</i> que será convertida para caracteres maiúsculos.
----------------	---

#### Exemplo:

```
cTexto := "ADVPL"  
  
MSGINFO("Texto:"+LOWER(cTexto))
```

## Manipulação de data / hora

---

## CDOW()

---

Função que converte uma data para uma cadeia de caracteres.

#### Sintaxe: CDOW( *dExp* )

#### Parâmetros:

<b>dExp</b>	Data que será convertida.
-------------	---------------------------

#### Retorno:

<b>cDayWeek</b>	Nome do dia da semana como uma cadeia de caracteres. A primeira letra é maiúscula e as demais minúsculas.
-----------------	---

**Exemplo:**

```
dData := DATE() // Resultado: 09/01/90  
cDiaDaSemana := CDOW(DATE()) // Resultado: Friday  
cDiaDaSemana := CDOW(DATE() + 7) // Resultado: Friday  
cDiaDaSemana := CDOW(CTOD("06/12/90")) // Resultado: Tuesday
```

**CMONTH()**

---

Função de conversão de datas que retorna uma cadeia de caracteres com o nome do mês em inglês.

**Sintaxe: CMONTH( *dData* )**

**Parâmetros:**

<b>dData</b>	Data que será convertida.
--------------	---------------------------

**Retorno:**

<b>cMonth</b>	Retorna o nome do mês em uma cadeia de caracteres. A primeira letra do retorno em maiúscula e o restante do nome, em minúsculas.
---------------	--

**Exemplo:**

```
cMes := CMONTH(DATE()) // Resultado: September  
cMes := CMONTH(DATE() + 45) // Resultado: October  
cMes := CMONTH(CTOD("12/01/94")) // Resultado: December  
cMes := SUBSTR(CMONTH(DATE()), 1, 3) + STR(DAY(DATE())) //  
Resultado: Sep 1
```

**DATE()**

---

Função que retorna a data do atual sistema. O formato de saída é controlado pelo comando SET DATE, sendo que o formato padrão é mm/dd/yy.

**Sintaxe: DATE()**

**Parâmetros:**

<b>Nenhum</b>	()
---------------	----

**Retorno:**

<b>dData</b>	Data do sistema.
--------------	------------------

**Exemplo:**

```
dData := DATE() // Resultado: 09/01/01  
dData := DATE() + 30 // Resultado: 10/01/01  
dData := DATE() - 30 // Resultado: 08/02/90  
dData := DATE()  
cMes := CMONTH(dData) // Resultado: September
```

**DAY()**

Função de conversão de datas usada para converter o valor data em um número inteiro que representa o dia do mês. Esta função pode ser usada em conjunto com CMONTH() e YEAR() para formatar datas. Pode ser usada também em diversos cálculos envolvendo datas.

**Sintaxe:** DAY( *dData* )

**Parâmetros:**

<b>dData</b>	Data que será convertida.
--------------	---------------------------

**Retorno:**

<b>nDias</b>	Se o mês do argumento dData for fevereiro, anos bissextos são considerados. Se a data do argumento dData for 29 de fevereiro e o ano não for bissexto, ou se o argumento dData for vazio.
--------------	---

**Exemplo:**

```
// Estes exemplos mostram a função DAY() de diversas maneiras:  
dData := DATE() // Resultado: 09/01/01  
nDia := DAY(DATE()) // Resultado: 1  
nDia := DAY(DATE()) + 1 // Resultado: 2  
nDia := DAY(CTOD("12/01/94")) // Resultado: 1  
// Este exemplo mostra a função DAY() usada em conjunto com CMONTH()  
e  
YEAR() para formatar o valor da data:  
dData := Date()  
cData := CMONTH(dData) + STR(DAY(dData)) + "," + STR(YEAR(dData)) //  
Resultado: June 15, 2001
```

## DOW()

---

Função que converte uma data para o valor numérico que representa o dia da semana. Útil quando se deseja fazer cálculos semanais. DOW() é similar a CDOW(), que retorna o dia da semana como uma cadeia de caracteres.

**Sintaxe:** DOW( *dData* )

**Parâmetros:**

<b>dData</b>	Data que será convertida.
--------------	---------------------------

**Retorno:**

<b>nDia</b>	Retorna um número entre zero e sete, representando o dia da semana. O primeiro dia da semana é 1 (Domingo) e o último é 7 (Sábado). Se a data for vazia ou inválida, DOW() retorna zero.
-------------	--

**Exemplo:**

```
dData := DATE() // Resultado: 09/01/01
nDiaDaSemana := DOW(DATE()) // Resultado: 3
cDiaDaSemana := CDOW(DATE()) // Resultado: Tuesday
nDiaDaSemana := DOW(DATE() - 2) // Resultado: 1
cDiaDaSemana := CDOW(DATE() - 2) // Resultado: Sunday
```

## DTOC()

---

Função para conversão de uma data para uma cadeia de caracteres formatada segundo o padrão corrente, definido pelo comando SET DATE. Se for necessária a utilização de formatação especial, use a função TRANSFORM().

Em expressões de índices de arquivo, use DTOS() no lugar de DTOC() para converter datas para cadeia de caracteres.

**Sintaxe:** DTOC( *dData* )

**Parâmetros:**

<b>dData</b>	Data que será convertida.
--------------	---------------------------

**Retorno:**

<b>cData</b>	É uma cadeia de caracteres representando o valor da data. O retorno é formatado utilizando-se o formato corrente definido pelo comando SET DATE FORMAT. O formato padrão é mm/dd/yy. Para uma data nula ou inválida, o retorno será uma cadeia de caracteres com espaços e tamanho igual ao formato atual.
--------------	--

**Exemplo:**

```
cData := DATE() // Resultado: 09/01/90  
cData := DTOC(DATE()) // Resultado: 09/01/90  
cData := "Today is " + DTOC(DATE()) // Resultado: Today is 09/01/90
```

**DTOS()**

---

Função para conversão de uma data que pode ser usada para criar expressões de índice. O resultado é estruturado visando manter a ordem correta do índice (ano, mês, dia).

**Sintaxe: DTOS( *dData* )**

**Parâmetros:**

<b>dData</b>	Data que será convertida.
--------------	---------------------------

**Retorno:**

<b>sData</b>	Retorna uma cadeia de caracteres com oito bytes de tamanho no formato yyyyymmdd. Quando <i>dData</i> é nulo ou invalido, DTOS() retorna uma cadeia de caracteres com oito espaços. O valor retornado não é afetado pela formatação da data corrente.
--------------	--

**Exemplo:**

```
cData := DATE() // Resultado: 09/01/90  
cData := DTOS(DATE()) // Resultado: 19900901  
nLen := LEN(DTOS(CTOD(""))) // Resultado: 8
```

## **ELAPTIME()**

---

Função que retorna uma cadeia de caracteres contendo a diferença de tempo no formato hh:mm:ss, onde hh é a hora ( 1 a 24 ), mm os minutos e ss os segundos.

**Sintaxe:** ElapTime( *cHoraInicial* , *cHoraFinal* )

**Parâmetros:**

cHoraInicial	Informe a hora inicial no formato hh:mm:ss, onde hh é a hora ( 1 a 24 ), mm os minutos e ss os segundos.
CHoraFinal	Informe a hora final no formato hh:mm:ss, onde hh é a hora ( 1 a 24 ), mm os minutos e ss os segundos.

**Retorno:**

Caracter	A diferença de tempo no formato hh:mm:ss, onde hh é a hora ( 1 a 24 ), mm os minutos e ss os segundos.
----------	--

**Exemplo:**

```
cHoraInicio := TIME() // Resultado: 10:00:00  
...  
<instruções>  
...  
cElapsed := ELAPTIME(TIME(), cHoraInicio)
```

## **MONTH()**

---

Função de conversão que extrai da data o valor numérico do mês, semelhante a função que retorna o nome do mês a partir do valor de dData.

**Sintaxe:** MONTH( *dData* )

**Parâmetros:**

<b>dData</b>	Data que será convertida.
--------------	---------------------------

**Retorno:**

<b>Numérico</b>	>=0 e <=12 ☐ Para uma data válida. 0 ☐ Se a data for nula ou inválida.
-----------------	---

**Exemplo:**

```
dData := DATE() // Resultado: 09/01/01  
nMes := MONTH(DATE()) // Resultado: 9  
nMes := MONTH(DATE()) + 1 // Resultado: 10
```

**SECONDS()**

---

Esta função retorna o número de segundos decorridos desde a meia-noite, segundo a hora do sistema. Está relacionada à função TIME() que retorna a hora do sistema como uma cadeia de caracteres no formato hh:mm:ss.

**Sintaxe: SECONDS()****Parâmetros:**

<b>Nenhum</b>	()
---------------	----

**Retorno:**

<b>Numérico</b>	>=0 e <=86399    Retorna a hora do sistema em segundos. O valor numérico representa o número de segundos decorridos desde a meia-noite, baseado no relógio de 24 horas e varia de 0 a 86399.
-----------------	--

**Exemplo:**

```
cHora := TIME() // Resultado: 10:00:00  
cSegundos := SECONDS() // Resultado: 36000.00  
  
//Este exemplo usa a função SECONDS() para cronometrar o tempo decorrido:  
  
LOCAL nStart, nElapsed  
nStart:= SECONDS()
```

## TIME()

---

Função que retorna a hora do sistema como uma cadeia de caracteres, e que está relacionada com SECONDS(), que retorna o valor inteiro representando o número de segundos desde a meia-noite. SECONDS() é geralmente usada no lugar de TIME() para cálculos.

**Sintaxe:** TIME()

**Parâmetros:**

Nenhum	.
--------	---

**Retorno:**

<b>Caracter</b>	A hora do sistema como uma cadeia de caracteres no formato hh:mm:ss onde hh é a hora ( 1 a 24 ), mm os minutos e ss os segundos.
-----------------	--

**Exemplo:**

```
cTime := TIME() // Resultado: 10:37:17  
cHora := SUBSTR(cTime, 1, 2) // Resultado: 10  
cMinutos := SUBSTR(cTime, 4, 2) // Resultado: 37  
cSegundos := SUBSTR(cTime, 7, 2) // Resultado: 17
```

## YEAR()

---

YEAR() é uma função de conversão de data que extrai o valor numérico do ano. YEAR() é membro de um grupo de funções que retornam valores numéricos de uma data. O grupo inclui DAY() e MONTH() que retornam o dia e o mês como valores numéricos.

**Sintaxe:** YEAR( *dData* )

**Parâmetros:**

<b>dData</b>	Data que será convertida.
--------------	---------------------------

**Retorno:**

<b>Numérico</b>	Valor numérico do ano da data especificada em <i>dData</i> incluindo os dígitos do século. O valor retornado não é afetado pelos valores especificados pelos comandos SET DATE ou SET CENTURY.  Para uma data inválida ou nula será retornado o valor 0.
-----------------	--

**Exemplo 01:**

```
dData := DATE() // Resultado: 09/20/01  
dAno := YEAR(dData) // Resultado: 2001  
dAno := YEAR(dData) + 11 // Resultado: 2012
```

**Exemplo 02:**

```
// Este exemplo cria uma função de usuário que usa a função YEAR()  
para formatar o valor da  
// data:  
  
cData := Mdy(DATE()) // Result: September 20, 1990  
FUNCTION Mdy( dDate )  
RETURN CMONTH(dDate) + " " + LTRIM(STR(DAY(dDate))) + "," +  
STR(YEAR(dDate))
```

## Manipulação de variáveis numéricas

### ABS()

Retorna um valor absoluto (independente do sinal) com base no valor especificado no parâmetro.

**Sintaxe: ABS(nValor)**

**Parâmetros**

<b>nValor</b>	Valor que será avaliado
---------------	-------------------------

**Exemplo:**

```
nPessoas := 20  
nLugares := 18  
  
IF nPessoas < nLugares  
    MSGINFO("Existem "+CVALTOCHAR(nLugares-  
nPessoas)+" disponíveis")  
ELSE  
    MSGSTOP("Existem "+CVALTOCHAR(ABS(nLugares-  
nPessoas))+ "faltando")  
ENDIF
```

## ALEATORIO()

Gera um número aleatório de acordo com a semente passada. Esta função retorna um número aleatório menor ou igual ao primeiro parâmetro informado, usando como semente o segundo parâmetro. É recomendado que esta semente seja sempre o último número aleatório gerado por esta função.

**Sintaxe:** Aleatorio(nMax,nSeed)

**Parâmetros**

<b>nMax</b>	Número máximo para a geração do número aleatório.
<b>nSeed</b>	Semente para a geração do número aleatório.

**Exemplo – Função ALEATORIO()**

```
nSeed := 0
For i := 1 to 100
nSeed := Aleatorio(100,nSeed)
? Str(i,3)+"§ numero aleatorio gerado: "+Str(nSeed,3)
Next i
inkey(0)
Return
```

## INT()

Retorna a parte inteira de um valor especificado no parâmetro.

**Sintaxe:** INT(nValor)

**Parâmetros**

<b>nValor</b>	Valor que será avaliado.
---------------	--------------------------

### **Exemplo:**

```
STATIC FUNCTION COMPRAR(nQuantidade)

LOCAL nDinheiro := 0.30
LOCAL nPrcUnit := 0.25

IF nDinheiro >= (nQuantidade*nPrcUnit)
    RETURN nQuantidade
ELSEIF nDinheiro > nPrcUnit
    nQuantidade := INT(nDinheiro / nPrcUnit)
ELSE
    nQuantidade := 0
ENDIF

RETURN nQuantidade
```

### **NOROUND()**

Retorna um valor, truncando a parte decimal do valor especificado no parâmetro de acordo com a quantidade de casas decimais solicitadas.

**Sintaxe: NOROUND(nValor, nCasas)**

**Parâmetros**

<b>nValor</b>	Valor que será avaliado.
<b>nCasas</b>	Número de casas decimais válidas. A partir da casa decimal especificada os valores serão desconsiderados.

### **Exemplo – Função NOROUND()**

```
nBase := 2.985
nValor := NOROUND(nBase,2)      ↗ 2.98
```

### **RANDOMIZE()**

Através da função RANDOMIZE() , geramos um numero inteiro aleatório, compreendido entre a faixa inferior e superior recebida através dos parâmetros nMinimo e nMaximo, respectivamente.

### **Observação :**

- O limite inferior recebido através do parâmetro nMinimo é "maior ou igual a ", podendo ser sorteado e fazer parte do retorno; porém o limite superior é "menor que", de modo a nunca será atingido ou devolvido no resultado. Por exemplo , a chamada da função RANDOMIZE(1,2) sempre retornará 1 .

**Sintaxe: RANDOMIZE ( < nMinimo > , < nMaximo > )**

#### **Parâmetros**

<b>nMinimo</b>	Corresponde ao menor numero a ser gerado pela função.
<b>nMaximo</b>	Corresponde ao maior número ( menos um ) a ser gerado pela função.

#### **Retorno:**

<b>Numérico</b>	Número randômico , compreendido no intervalo entre (nMinimo) e (nMaximo-1) : O número gerado pode ser maior ou igual à nMinimo e menor ou igual a nMaximo-1 .
-----------------	--

### **ROUND()**

---

Retorna um valor, arredondando a parte decimal do valor especificado no parâmetro de acordo com a quantidades de casas decimais solicitadas, utilizando o critério matemático.

**Sintaxe: ROUND(nValor, nCasas)**

#### **Parâmetros**

<b>nValor</b>	Valor que será avaliado.
<b>nCasas</b>	Número de casas decimais válidas. As demais casas decimais sofrerão o arredondamento matemático, aonde:  Se $nX \leq 4 \rightarrow 0$ , senão +1 para a casa decimal superior.

#### **Exemplo:**

```
nBase := 2.985  
nValor := ROUND(nBase,2)      → 2.99
```

## Manipulação de arquivos

### **ADIR()**

---

Função que preenche os arrays passados com os dados dos arquivos encontrados, através da máscara informada. Tanto arquivos locais (Remote) como do servidor podem ser informados.

**Importante:** *ADir é uma função obsoleta, utilize sempre Directory().*

**Sintaxe:** **ADIR([ cArqEspec ], [ aNomeArq ], [ aTamanho ], [ aData ], [ aHora ], [ aAtributo ] )**

**Parâmetros:**

<b>cArqEspec</b>	Caminho dos arquivos a serem incluídos na busca de informações. Segue o padrão para especificação de arquivos, aceitando arquivos no servidor Protheus e no Cliente. Caracteres como * e ? são aceitos normalmente. Caso seja omitido, serão aceitos todos os arquivos do diretório default ( *.* ).
<b>aNomeArq</b>	Array de Caracteres. É o array com os nomes dos arquivos encontrados na busca. O conteúdo anterior do array é apagado.
<b>aTamanho</b>	Array Numérico. São os tamanhos dos arquivos encontrados na busca.
<b>aData</b>	Array de Datas. São as datas de modificação dos arquivos encontrados na busca.
<b>aHora</b>	Array de Caracteres. São os horários de modificação dos arquivos encontrados. Cada elemento contém horário no formato: hh:mm:ss.
<b>aAtributos</b>	Array de Caracteres. São os atributos dos arquivos, caso esse array seja passado como parâmetros, serão incluídos os arquivos com atributos de sistema e ocultos.

**Retorno:**

<b>nArquivos</b>	Quantidade de arquivos encontrados.
------------------	-------------------------------------

**Exemplo:**

```
LOCAL aFiles[ADIR("*.TXT")]
ADIR("*.TXT", aFiles)
AEVAL(aFiles, { |element| QOUT(element) })
```

## **CGETFILE()**

---

Função utilizada para seleção de um arquivo ou diretório, disponibilizando uma interface gráfica e amigável para o usuário. Esta função está normalmente associada ao recurso de abrir ou salvar arquivos, permitindo a digitação opcional do nome do arquivo que será gravado.

**Sintaxe:** cGetFile ( **ExpC1**, **ExpC2**, **ExpN1**, **ExpC3**, **ExpL1**, **ExpN2**,**ExpL2** )

### **Parâmetros:**

<b>ExpC1</b>	Máscara para filtro (Ex: 'Informes Protheus (*.##R)   *.##R').
<b>ExpC2</b>	Título da Janela.
<b>ExpN1</b>	Número da máscara default ( Ex: 1 p/ *.exe ).
<b>ExpC3</b>	Diretório inicial se necessário.
<b>ExpL1</b>	.T. para mostrar botão como 'Salvar' e .F. para botão 'Abrir'.
<b>ExpN2</b>	Máscara de bits para escolher as opções de visualização do Objeto.
<b>ExpL2</b>	.T. para exibir diretório [Servidor] e .F. para não exibir.

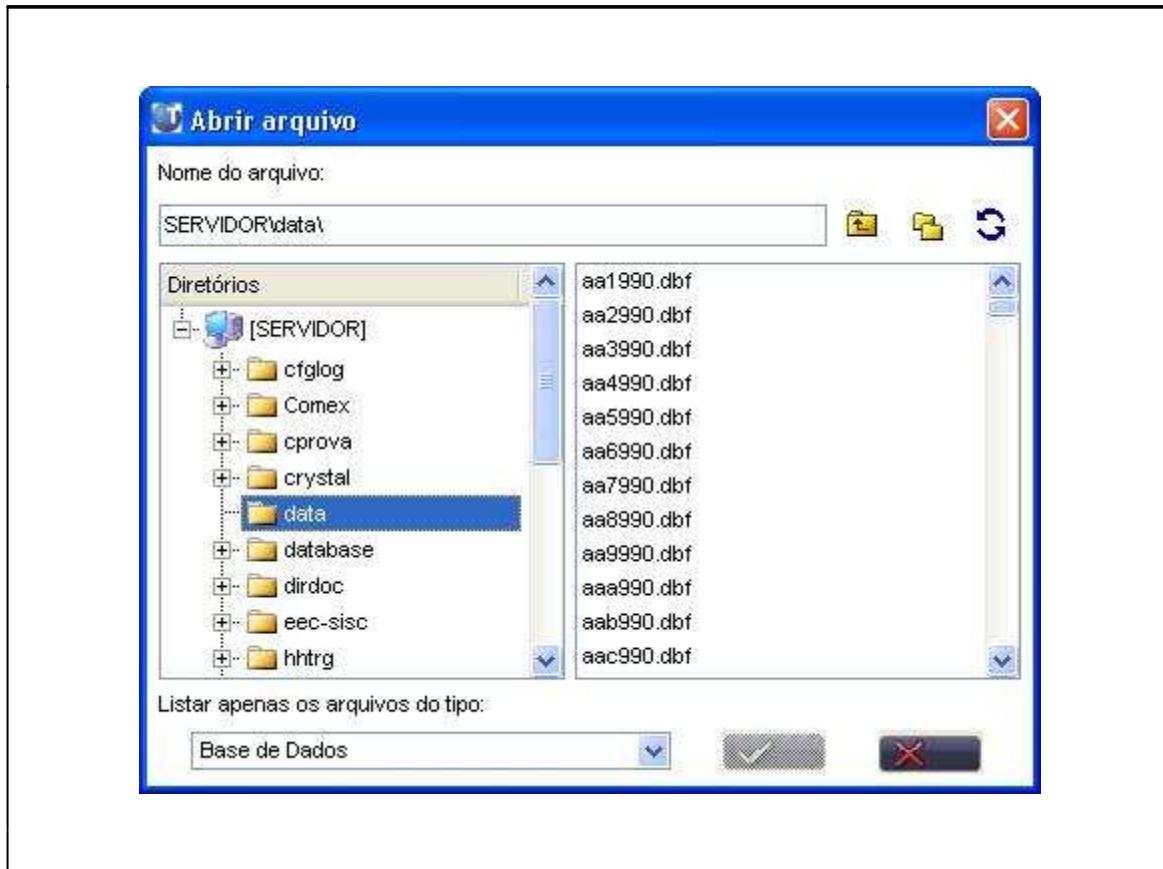
### **Máscaras de bits para opções:**

<b>GETF_OVERWRITEPROMPT</b>	Solicita confirmação para sobreescriver.
<b>GETF_MULTISELECT</b>	Permite selecionar múltiplos arquivos.
<b>GETF_NOCHANGEDIR</b>	Não permite mudar o diretório inicial.
<b>GETF_LOCALFLOPPY</b>	Exibe o(s) Drive(s) de disquete da máquina local.
<b>GETF_LOCALHARD</b>	Exibe o(s) HardDisk(s) Local(is).
<b>GETF_NETWORKDRIVE</b>	Exibe os drives da rede ( Mapeamentos ).
<b>GETF_SHAREWARE</b>	Não implementado.
<b>GETF_RETDIRECTORY</b>	Retorna um diretório.

### **Exemplo:**

```
cGetFile ( '*.PRW|*.CH' , 'Fontes' , 1 , 'C:\VER507' , .F. ,
GETF_LOCALHARD + GETF_LOCALFLOPPY)
```

**Aparência:**



---

Para permitir a seleção de diversos arquivos contidos em um diretório é necessário combinar as funções CGETFILE(), DIRECTORY() e o objeto LISTBOX() conforme abaixo:



*Dica*

- CGETFILE: exibe os diretórios disponíveis e retorna o nome do item selecionado.
  - DIRECTORY: efetua a leitura dos arquivos contidos no diretório retornado pela CGETFILE.
  - LISTBOX: Exibe uma tela de seleção de com a opção de marcação, para que sejam selecionados os arquivos que serão processados.
-

---

**Exemplo: Seleção de múltiplos arquivos com CGETFILE, DIRECTORY() e LISTBOX()**

---

**Função Principal: SELFILE()**

---

```
#include "protheus.ch"

//+-----+
//| Rotina | SELFILE | Autor | ARNALDO R. JUNIOR | Data |
01.01.2007 |
//+-----+
//| Descr. | Função exemplo para seleção de múltiplos arquivos.    |
//+-----+
//| Uso      | CURSO DE ADVPL                                |
//+-----+


USER FUNCTION SELFILE()

LOCAL cDirectory := ""
LOCAL aArquivos  := {}
LOCAL nArq     := 0

PRIVATE aParamFile:= ARRAY(1)

IF !PARBOXFILE()
    RETURN
ENDIF

// Exibe a estrutura de diretório e permite a seleção dos arquivos
que serão processados
cDirectory      := ALLTRIM(cGetFile("Arquivos de
Dados|" + aParamFile[1] + "|", 'Importação de lançamentos', 0, .T.,
GETF_OVERWRITEPROMPT + GETF_NETWORKDRIVE + GETF_RETDIRECTORY,.T.))
aArquivos      := Directory(cDirectory+"*.*")
aArquivos      := MARKFILE(aArquivos,cDirectory,aParamFile[1],@lSelecao)

FOR nArq TO Len(aArquivos)

    IF !aArquivos[nArq][1]
        LOOP
    ENDIF

    <...processamento...>

NEXT nArq

RETURN
```

## Função auxiliar: PARBOXFILE()

```
//+-----+
//| Rotina | PARBOXFILE | Autor | ARNALDO R. JUNIOR  Data |
01.01.2007 |
//+-----+
//| Descr. | Função exemplo de uso da PARAMBOX em conjunto com
CGETFILE|
//+-----+
//| Uso      | CURSO DE ADVPL
//+-----+
|
```

STATIC FUNCTION PARBOXFILE()

Local aParamBox := {}  
Local cTitulo := "Filtros Adicionais"  
Local aRet := {}  
Local bOk := {|| .T.}  
Local aButtons := {}  
Local lCentered := .T.  
Local nPosx  
Local nPosy  
Local cLoad := ""  
Local lCanSave := .F.  
Local lUserSave := .F.  
Local nX := 0  
Local lRet := .T.

AADD(aParamBox,{2,"Tipo de arquivo"  
,2,"\*.dbf","\*.dtc"},100,"AllwaysTrue()",.T.)  
lRet := ParamBox(aParamBox, cTitulo, aRet, bOk, aButtons, lCentered,  
nPosx, nPosy,, cLoad, lCanSave, lUserSave)  
IF ValType(aRet) == "A" .AND. Len(aRet) == Len(aParamBox)  
 For nX := 1 to Len(aParamBox)  
 If aParamBox[nX][1] == 1  
 aParam102[nX] := aRet[nX]  
 Elseif aParamBox[nX][1] == 2 .AND. ValType(aRet[nX]) ==  
"C"  
 aParam102[nX] := aRet[nX] // Tipo do arquivo  
 Elseif aParamBox[nX][1] == 2 .AND. ValType(aRet[nX]) ==  
"N"  
 aParam102[nX] := aParamBox[nX][4][aRet[nX]] // Tipo do  
arquivo  
 Endif  
 Next nX  
ENDIF  
RETURN lRet

## Função auxiliar: MARKFILE()

```
//+-----+
//| Rotina | MARKFILE | Autor | ARNALDO R. JUNIOR | Data |
01.01.2007 |
//+-----+
//| Descr. | Função exemplo para marcação de múltiplos arquivos. |
//+-----+
//| Uso    | CURSO DE ADVPL |
//+-----+

STATIC FUNCTION MARKFILE(aArquivos,cDiretorio,cDriver,lSelecao)

Local aChaveArq := {}
Local cTitulo      := "Arquivos para importação: "
Local bCondicao := {|| .T.}
// Variáveis utilizadas na seleção de categorias
Local oChkQual,IQual,oQual,cVarQ
// Carrega bitmaps
Local oOk       := LoadBitmap( GetResources(), "LBOK")
Local oNo       := LoadBitmap( GetResources(), "LBNO")
// Variáveis utilizadas para lista de filiais
Local nx       := 0
Local nAchou   := 0

//+-----+
//| Carrega os arquivos do diretório no array da ListBox |
//+-----+
For nX := 1 to Len(aArquivos)
    //+-----+
    //| aChaveArq - Contem os arquivos que serão exibidos para
seleção |
    //+-----+
    AADD(aChaveArq,{.F.,aArquivos[nX][1],cDiretorio})
Next nX

//+-----+
//| Monta tela para seleção dos arquivos contidos no diretório
|
//+-----+
DEFINE MSDIALOG oDlg TITLE cTitulo STYLE DS_MODALFRAME From 145,0 To
445,628;
OF oMainWnd PIXEL
oDlg:lEscClose := .F.
@ 05,15 TO 125,300 LABEL UPPER(cDriver) OF oDlg  PIXEL
@ 15,20 CHECKBOX oChkQual VAR IQual PROMPT "Inverte Seleção" SIZE
50, 10;
OF oDlg PIXEL;
```

```

ON CLICK (AEval(aChaveArq, {|z| z[1] := If(z[1]==.T.,.F.,.T.)});;
oQual:Refresh(.F.))
@ 30,20 LISTBOX oQual VAR cVarQ Fields HEADER
","", "Código", "Descrição" SIZE;
273,090 ON DBLCLICK
(aChaveArq:=Troca(oQual:nAt,aChaveArq),oQual:Refresh());
NoScroll OF oDlg PIXEL
oQual:SetArray(aChaveArq)
oQual:bLine := { | | {If(aChaveArq[oQual:nAt,1],oOk,oNo),;
aChaveArq[oQual:nAt,2],aChaveArq[oQual:nAt,3]}}
DEFINE SBUTTON FROM 134,240 TYPE 1 ACTION IIF(MarcaOk(aChaveArq),;
(ISelecao := .T., oDlg:End(),.T.),.F.) ENABLE OF oDlg
DEFINE SBUTTON FROM 134,270 TYPE 2 ACTION (ISelecao := .F.,
oDlg:End());
ENABLE OF oDlg
ACTIVATE MSDIALOG oDlg CENTERED

```

RETURN aChaveArq

#### **Função auxiliar: TROCA()**

```

//+-----+
//| Rotina | TROCA      | Autor | ARNALDO R. JUNIOR | Data |
01.01.2007 |
//+-----+
//| Uso     | CURSO DE ADVPL
|
//+-----+

```

```

STATIC FUNCTION Troca(nIt,aArray)
aArray[nIt,1] := !aArray[nIt,1]
Return aArray

```

## Função auxiliar: MARCAOK()

```
//+-----+
//| Rotina | MARCAOK   | Autor | ARNALDO R. JUNIOR | Data |
01.01.2007 |
//+-----+
//| Uso    | CURSO DE ADVPL           |
//+-----+
STATIC FUNCTION MarcaOk(aArray)
Local IRet:=.F.
Local nx:=0

// Checa marcações efetuadas
For nx:=1 To Len(aArray)
    If aArray[nx,1]
        IRet:=.T.
    EndIf
Next nx
// Checa se existe algum item marcado na confirmação
If !IRet
    HELP("SELFILE",1,"HELP","SEL. FILE","Não existem itens marcados",1,0)
EndIf

Return IRet
```

## CPYS2T()

Função utilizada para copiar um arquivo do servidor para o cliente (Remote), sendo que os caracteres “\*” e “?” são aceitos normalmente. Caso a compactação seja habilitada (*ICompacta*), os dados serão transmitidos de maneira compacta e descompactados antes do uso.

**Sintaxe:** CPYS2T ( < cOrigem > , < cDestino > , [ ICompacta ] )

**Parâmetros:**

<b>cOrigem</b>	Nome(s) dos arquivos a serem copiados, aceita apenas arquivos no servidor, WildCards ( * e ? ) são aceitos normalmente.
<b>cDestino</b>	Diretório com o destino dos arquivos no Client ( Remote ).
<b>ICompacta</b>	Indica se a cópia deve ser feita compactando o arquivo antes do envio.

**Retorno:**

<b>Lógico</b>	ISucess retorna .T. caso o arquivo seja copiado com sucesso, ou .F. em caso de falha na cópia.
---------------	--

**Exemplo:**

```
// Copia arquivos do servidor para o remote local, compactando antes  
de transmitir  
CpyS2T( "\BKP\MANUAL.DOC", "C:\TEMP", .T. )  
// Copia arquivos do servidor para o remote local, sem compactar  
antes de transmitir  
CpyS2T( "\BKP\MANUAL.DOC", "C:\TEMP", .F. )
```

**CPYT2S()**

Função utilizada para copiar um arquivo do cliente (Remote) para o servidor, sendo que os caracteres “\*” e “?” são aceitos normalmente. Caso a compactação seja habilitada (*ICompacta*), os dados serão transmitidos de maneira compacta e descompactados antes do uso.

Sintaxe: CpyT2S( **cOrigem**, **cDestino**, [ **ICompacta** ] )

**Parâmetros:**

<b>cOrigem</b>	Nomes dos arquivos a serem copiados, aceita apenas arquivos locais (Cliente), WildCards são aceitos normalmente.
<b>cDestino</b>	Diretório com o destino dos arquivos no remote ( Cliente ).
<b>ICompacta</b>	Indica se a cópia deve ser feita compactando o arquivo antes.

**Retorno:**

<b>Lógico</b>	Indica se o arquivo foi copiado para o cliente com sucesso.
---------------	---

**Exemplo:**

```
// Copia arquivos do cliente( remote ) para o Servidor compactando  
antes de transmitir  
CpyT2S( "C:\TEMP\MANUAL.DOC", "\BKP", .T. )  
// Copia arquivos do cliente( remote ) para o Servidor sem  
compactar.  
CpyT2S( "C:\TEMP\MANUAL.DOC", "\BKP" )
```

## CURDIR()

---

Função que retorna o diretório corrente do servidor. O caminho retornado é sempre relativo ao RootPath definido na configuração do Environment no .INI do Protheus Server. Inicialmente, o diretório atual da aplicação é o constante na chave StartPath, também definido na configuração do Environment no .INI do Protheus Server.

Caso seja passado o parâmetro cNovoPath, este path é assumido como sendo o pacote atual. Caso o path recebido como parâmetro não exista, seja inválido, ou seja um pacote absoluto (iniciado com uma letra de drive ou caminho de rede), a função não irá setar o novo pacote, mantendo o atual.

**Sintaxe:** CURDIR ( [ cNovoPath ] )

**Parâmetros:**

<b>cNovoPath</b>	Caminho relativo, com o novo diretório que será ajustado como corrente.
------------------	---

**Retorno:**

<b>Caracter</b>	Diretório corrente, sem a primeira barra.
-----------------	---

**Exemplo:**

```
cOldDir := curdir()
cNewDir := '\webadv\xis'
curdir(cNewDir) // Troca o path
If cNewDir <> '\'+curdir() // E verifica se trocou mesmo
    conout('Falha ao Trocar de Path de '+cOldDir + ' para '+cNewDir)
Else
    conout('Path de '+cOldDir + ' trocado para '+cNewDir+ ' com
    sucesso.')
Endif
```

## DIRECTORY()

---

Função de tratamento de ambiente que retorna informações a respeito dos arquivos no diretório corrente ou especificado. É semelhante a ADIR(), porém retorna um único vetor ao invés de adicionar valores a uma série de vetores existentes passados por referência.

DIRECTORY() pode ser utilizada para realizar operações em conjuntos de arquivos. Em combinação com AEVAL(), você pode definir um bloco que pode ser aplicado a todos os arquivos que atendam a <cDirSpec> especificada.

Para tornar as referências aos vários elementos de cada sub-vetor de arquivo mais

legíveis, é fornecido o arquivo *header Directry.ch*, que contém os #defines para os *subarray subscripts*.

**TABELA A: Atributos de DIRECTORY()**

Atributo	Significado
H	Incluir arquivos ocultos
S	Incluir arquivos de sistema
D	Incluir diretórios
V	Procura pelo volume DOS e exclui outros arquivos

*Nota: Arquivos normais são sempre incluídos na pesquisa, a não ser que V seja especificado.*

**TABELA B: Estrutura dos Subvetores de DIRECTORY()**

Posição	Metasímbolo	Directry.ch
1	cNome	F_NAME
2	cTamanho	F_SIZE
3	dData	F_DATE
4	cHora	F_TIME
5	cAtributos	F_ATT

**Sintaxe: DIRECTORY ( < cDirSpec > , [ ] )**

#### Parâmetros:

<b>cDirSpec</b>	<cDirSpec> especifica o <i>driver</i> , diretório e arquivo para a pesquisa no diretório. Caracteres do tipo coringa são permitidos na especificação de arquivos. Caso <cDirSpec> seja omitido, o valor padrão é <i>*.*</i> . O caminho especificado pode estar na estação (remote), ou no servidor, obedecendo às definições de Path Absoluto/Relativo de acesso.
<b>cAtributos&gt;</b>	<cAtributos> especifica que arquivos com atributos especiais devem ser incluídos na informação retornada. <cAtributos> consiste em uma cadeia de caracteres que contém um ou mais dos seguintes caracteres, contidos na tabela adicional A , especificada anteriormente.

#### Retorno:

<b>Array</b>	DIRECTORY() retorna um vetor de sub-vetores, sendo que cada sub-vetor contém informações sobre cada arquivo que atenda a <cDirSpec>. Veja maiores detalhes na Tabela B, discriminada anteriormente.
--------------	---

**Exemplo:**

```
#INCLUDE "Directry.ch"

aDirectory := DIRECTORY("*.*","D")
AEVAL( aDirectory, {|aFile| CONOUT(aFile[F_NAME])} )
```

**DIRREMOVE()**

Função que elimina um diretório específico. Caso especifiquemos um pacote sem a unidade de disco , ele será considerado no ambiente do Servidor , a partir do RootPath do ambiente ( caso o path comece com \ ), ou a partir do diretório corrente (caso o path não seja iniciado com \ ).

Quando especificado um pacote(path) absoluto ( com unidade de disco preenchida ), a função será executada na estação onde está sendo executado o Protheus Remote. Quando executamos a função DirRemove() em JOB ( processo isolado no Server, sem interface ), não é possível especificar um Path absoluto de disco. Caso isto seja realizado, a função retornará .F. e FError() retornará -1 ( Syntax Error ).

Note que é necessário ter direitos suficientes para remover um diretório, e o diretório a ser eliminado precisa estar vazio, sem subdiretórios ou arquivos dentro do mesmo.

**Sintaxe: DIRREMOVE (< cDiretorio >)****Parâmetros:**

<b>cDiretorio</b>	Nome do diretório a ser removido.
-------------------	-----------------------------------

**Retorno:**

<b>Lógico</b>	ISucesso será .T. caso o diretório tenha sido eliminado, ou .F. caso não seja possível excluir o diretório. Quando a função DirRemove retornar .F., é possível obter mais detalhes da ocorrência recuperando o código do Erro através da função FError().
---------------	---

**Exemplo:**

```
cDelPath := 'c:\TmpFiles'
!RemoveOk := DIRREMOVE(cDelPath)

IF !RemoveOk
    MsgStop('Falha ao remover a pasta '+cDelPath+' ( File Error
'+str(Fewrror(),4)+' ) ')
Else
    MsgStop('Pasta '+cDelPath+' removida com sucesso.')
Endif
```

## **DISKSPACE()**

---

Função de ambiente que determina quantos *bytes* estão disponíveis em uma determinada unidade de disco. Esta função obtém a informação sempre relativa à estação onde está sendo executado o Protheus Remote. Através do parâmetro nDrive , selecionamos qual a unidade de disco que desejamos obter a informação do espaço livre, onde:

- 0 : Unidade de disco atual da estação (DEFAULT).**
- 1 : Drive A: da estação remota.**
- 2 : Drive B: da estação remota.**
- 3 : Drive C: da estação remota.**
- 4 : Drive D: da estação remota ... e assim por diante.**

Caso a função DiskSpace seja executada através de um *Job* (processo isolado no Servidor, sem interface Remota) , ou seja passado um argumento de unidade de disco inexistente ou indisponível, a função DISKSPACE() retornará -1

**Sintaxe: DISKSPACE ( [ nDrive ] )**

**Parâmetros:**

<b>nDrive</b>	Número do driver, onde 0 é o espaço na unidade de disco corrente, e 1 é o drive A: do cliente, 2 é o drive B: do cliente, etc..
---------------	---

**Retorno:**

<b>Numérico</b>	Número de <i>bytes</i> disponíveis no disco informado como parâmetro.
-----------------	---

**Exemplo:**

```
nBytesLocal := DISKSPACE( ) // Retorna o espaço disponível na  
unidade de disco local  
  
IF nBytesLocal < 1048576  
    MsgStop('Unidade de Disco local possui menos de 1 MB livre.')  
Else  
    MsgStop('Unidade de disco local possui '+str(nBytes_A,12)+' bytes  
livres.')  
Endif  
nBytes_A := DISKSPACE( 1 ) // Retorna o espaço disponível no drive  
A: local ( remote ).  
  
If nBytes_A == -1  
    MsgStop('Unidade A: não está disponível ou não há disco no  
Drive')  
Elseif nBytes_A < 8192
```

```

MsgStop('Não há espaço disponível no disco. Substitua o disco na
Unidade A:')
Else
    MsgStop('Unidade A: Verificada . '+str(nBytes_A,12)+' bytes
livres.')
Endif

```

## **EXISTDIR()**

---

Função utilizada para determinar se um pacote de diretório existe e é valido.

**Sintaxe: EXISTDIR (< cPath >)**

**Parâmetros:**

<b>cPath</b>	<i>String</i> contendo o diretório que será verificado, caso seja feita uma verificação a partir do server, devemos informar a partir do rootPath do Protheus, caso contrário devemos passar o pacote completo do diretório.
--------------	--

**Retorno:**

<b>Lógico</b>	Retorna se verdadeiro(.T.) caso o diretório solicitado exista, falso(.F.) caso contrário.
---------------	---

### **Exemplo 01: No server a partir do rootPath**

```
IRet := ExistDir('\teste')
```

### **Exemplo 02: No client, passando o FullPath**

```
IRet := ExistDir('c:\APO')
```

## **FCLOSE()**

---

Função de tratamento de arquivos de baixo nível utilizada para fechar arquivos binários e forçar que os respectivos *buffers* do DOS sejam escritos no disco. Caso a operação falhe, FCLOSE() retorna falso (.F.). FERROR() pode então ser usado para determinar a razão exata da falha. Por exemplo, ao tentar-se usar FCLOSE() com um *handle* (tratamento dado ao arquivo pelo sistema operacional) inválido retorna falso (.F.) e FERROR() retorna erro 6 do DOS, *invalid handle*. Consulte FERROR() para obter uma lista completa dos códigos de erro.

**Nota:** Esta função permite acesso de baixo nível aos arquivos e dispositivos do DOS. Ela deve ser utilizada com extremo cuidado e exige que se conheça a fundo o sistema operacional utilizado.

**Sintaxe: FCLOSE ( < nHandle > )**

**Parâmetros:**

<b>nHandle</b>	<i>Handle</i> do arquivo obtido previamente através de FOPEN() ou FCREATE().
----------------	--

**Retorno:**

<b>Lógico</b>	Retorna falso (.F.) se ocorre um erro enquanto os <i>buffers</i> estão sendo escritos; do contrário, retorna verdadeiro (.T.).
---------------	--

**Exemplo:**

```
#include "Fileio.ch"

nHandle := FCREATE("Testfile", FC_NORMAL)

If !FCLOSE(nHandle)
    conout( "Erro ao fechar arquivo, erro numero: ", FERROR() )
EndIf
```

## **FCREATE()**

Função de baixo-nível que permite a manipulação direta dos arquivos textos como binários. Ao ser executada FCREATE() cria um arquivo ou elimina o seu conteúdo, e retorna o handle (manipulador) do arquivo, para ser usado nas demais funções de manutenção de arquivo. Após ser utilizado , o Arquivo deve ser fechado através da função

FCLOSE().

Na tabela abaixo , estão descritos os atributos para criação do arquivo, definidos no arquivo header fileio.ch.

**Atributos definidos no include FileIO.ch**

Constante	Valor	Descrição
FC_NORMAL	0	Criação normal do Arquivo (default/padrão).
FC_READONLY	1	Cria o arquivo protegido para gravação.
FC_HIDDEN	2	Cria o arquivo como oculto.
FC_SYSTEM	4	Cria o arquivo como sistema.

Caso desejemos especificar mais de um atributo, basta somá-los. Por exemplo , para criar um arquivo protegido contra gravação e escondido, passamos como atributo FC\_READONLY + FC\_HIDDEN..

**Nota:** Caso o arquivo já exista, o conteúdo do mesmo será ELIMINADO, e seu tamanho será truncado para 0 ( ZERO ) bytes.

**Sintaxe:** FCREATE ( < cArquivo > , [ nAtributo ] )

**Parâmetros:**

<b>cArquivo</b>	Nome do arquivo a ser criado, podendo ser especificado um pacote absoluto ou relativo, para criar arquivos no ambiente local (Remote) ou no Servidor, respectivamente.
<b>nAtributo</b>	Atributos do arquivo a ser criado (Vide Tabela de atributos abaixo). Caso não especificado, o DEFAULT é FC_NORMAL.

**Retorno:**

<b>Numérico</b>	A função retornará o <i>Handle</i> do arquivo para ser usado nas demais funções de manutenção de arquivo. O <i>Handle</i> será maior ou igual a zero. Caso não seja possível criar o arquivo, a função retornará o handle -1, e será possível obter maiores detalhes da ocorrência através da função FERROR().
-----------------	--

## FERASE()

---

Função utilizada para apagar um arquivo no disco. O Arquivo pode estar no Servidor ou na estação local (Remote). O arquivo para ser apagado deve estar fechado, não sendo permitido a utilização de caracteres coringa (wildcards).

**Sintaxe:** FERASE ( < cArquivo > )

**Parâmetros:**

<b>cArquivo</b>	Nome do arquivo a ser apagado . Pode ser especificado um path absoluto ou relativo , para apagar arquivos na estação local (Remote) ou no Servidor, respectivamente.
-----------------	--

**Retorno:**

<b>Numérico</b>	A função retornará 0 caso o arquivo seja apagado com sucesso, e -1 caso não seja possível apagar o arquivo. Caso a função retorne -1, é possível obter maiores detalhes da ocorrência através da função FERROR().
-----------------	---

**Exemplo:**

```
#include 'DIRECTRY.CH'

aEval(Directory("*.BAK"), { |aFile| FERASE(aFile[F_NAME]) })

// Este exemplo apaga um arquivo no cliente ( Remote ) , informando
o status da operação
IF FERASE("C:\ListaTXT.tmp") == -1
    MsgStop('Falha na deleção do Arquivo ( FError'+str(ferror(),4) +
')
Else
    MsgStop('Arquivo deletado com sucesso.')
ENDIF
```

**FILE()**

Função que verifica se existe um arquivo ou um padrão de arquivos, no diretório. Podem ser especificados caminhos absolutos (arquivos na estação - Remote) ou relativos (a partir do RootPath do Protheus Server), sendo os caracteres “\*” e “?” (wildcards) aceitos.

**Sintaxe: FILE ( < cArquivo > )**

**Parâmetros:**

<b>cArquivo</b>	Nome do arquivo, podendo ser especificado um pacote (caminho) . Caminhos locais (Remote) ou caminhos de servidor são aceitos, bem como (Caracteres “*” e “?” ).	<i>wildcards</i>
-----------------	---	------------------

**Retorno:**

<b>Lógico</b>	O retorno será .T. caso o arquivo especificado exista. Caso o mesmo não exista no path especificado, a função retorna .F.
---------------	---

**Exemplo:**

```
//Verifica no diretório corrente do servidor se existe o arquivo
teste.dbf
FILE("teste.dbf")

// Verifica no diretório Sigaadv do servidor se existe o arquivo
teste.dbf
FILE("\SIGAADV\TESTE.dbf")

// Verifica no diretório Temp do cliente (Remote) se existe o
arquivo teste.dbf
FILE("C:\TEMP\TESTE.dbf")
```



*Importante*

Caso a função FILE() seja executada em Job ( programa sem interface remota ), sendo passado um caminho absoluto de arquivo ( exemplo c:\teste.txt ) , a função retornará .F. e FERROR() retornará -1 ).

## FILENOEXT()

Função que retorna o nome de um arquivo contido em uma *string*, ignorando a extensão.

**Sintaxe:** FileNoExt( *cString* )

**Parâmetros**

<b>cString</b>	String contendo o nome do arquivo.
----------------	------------------------------------

**Exemplo:**

```
Local cString := '\SIGAADV\ARQZZZ.DBF'  
cString := FileNoExt( cString )  
// Retorno  ↴ "\SIGAADV\ARQZZZ"
```

## FOPEN()

Função de tratamento de arquivo de baixo nível que abre um arquivo binário existente para que este possa ser lido e escrito, dependendo do argumento <nModo>. Toda vez que houver um erro na abertura do arquivo, FERROR() pode ser usado para retornar o código de erro do Sistema Operacional. Por exemplo, caso o arquivo não exista, FOPEN() retorna -1 e FERROR() retorna 2 para indicar que o arquivo não foi encontrado. Veja FERROR() para uma lista completa dos códigos de erro.

Caso o arquivo especificado seja aberto, o valor retornado é o *handle* ( manipulador ) do Sistema Operacional para o arquivo. Este valor é semelhante a um alias no sistema de banco de dados, e ele é exigido para identificar o arquivo aberto para as outras funções de tratamento de arquivo. Portanto, é importante sempre atribuir o valor que foi retornado a uma variável para uso posterior, como mostra o exemplo desta função.

**Nota:** Esta função permite acesso de baixo nível a arquivos e dispositivos. Ela deve ser utilizada com extremo cuidado e exige que se conheça a fundo o sistema operacional utilizado.



**Importante**

- FOPEN procura o arquivo no diretório corrente e nos diretórios configurados na variável de pesquisa do Sistema Operacional, a não ser que um path seja declarado explicitamente como parte do argumento <cArq>.
- Por serem executadas em um ambiente cliente-servidor, as funções de tratamento de arquivos podem trabalhar em arquivos localizados no cliente (estação) ou no servidor. O ADVPL identifica o local onde o arquivo será manipulado através da existência ou não da letra do drive no nome do arquivo passado em <cArq>. Ou seja, se o arquivo for especificado com a letra do driver, será aberto na estação. Caso contrário, será aberto no servidor com o diretório configurado como rootpath sendo o diretório raiz para localização do arquivo.

#### Sintaxe: FOPEN ( < cArq > , [ nModo ] )

##### Parâmetros:

cArq	Nome do arquivo a ser aberto que inclui o pacote, caso haja um.
nModo	Modo de acesso DOS solicitado que indica como o arquivo aberto deve ser acessado. O acesso é de uma das categorias relacionadas na Tabela A e as restrições de compartilhamento relacionada na Tabela B. O modo padrão é zero, somente para leitura, com compartilhamento por Compatibilidade. Ao definirmos o modo de acesso, devemos somar um elemento da Tabela A com um elemento da Tabela B.

##### Retorno:

Numérico	FOPEN() retorna o handle de arquivo aberto na faixa de zero a 65.535. Caso ocorra um erro, FOPEN() retorna -1.
----------	--

##### Exemplo:

```
#include 'fileio.ch'
...
nH := fopen('\\sigaadv\\error.log' , FO_READWRITE + FO_SHARED )
If nH == -1
    MsgStop('Erro de abertura : FERROR '+str(ferror(),4))
Else
    MsgStop('Arquivo aberto com sucesso.')
    ...
    fclose(nH)
Endif
...
```

**Tabela A: Modos de acesso a arquivos binários**

Modo	Constate(fileio.ch)	Operação
0	FO_READ	Aberto para leitura (padrão assumido)
1	FO_WRITE	Aberto para gravação
2	FO_READWRITE	Aberto para leitura e gravação

**Tabela B: Modos de acesso de compartilhamento a arquivos binários**

Modo	Constate(fileio.ch)	Operação
0	FO_COMPAT	Modo de Compatibilidade (Default).
16	FO_EXCLUSIVE	Acesso total exclusivo.
32	FO_DENYWRITE	Acesso bloqueando a gravação de outros processos ao arquivo.
48	FO_DENYREAD	Acesso bloqueando a leitura de outros processos ao arquivo.
64	FO_DENYNONE	Acesso compartilhado. Permite a leitura e gravação por outros.

## FREAD()

---

Função que realiza a leitura dos dados a partir um arquivo aberto, através de FOPEN(), FCREATE() e/ou FOPENPORT(), e armazena os dados lidos por referência no buffer informado.

FREAD() lerá até o número de bytes informado em nQtdBytes; caso aconteça algum erro ou o arquivo chegue ao final, FREAD() retornará um número menor que o especificado em nQtdBytes. FREAD() lê normalmente caracteres de controle (ASC 128, ASC 0, etc.) e lê a partir da posição atual do ponteiro atual do arquivo, que pode ser ajustado ou modificado pelas funções FSEEK(), FWRITE() ou FREADSTR().

A variável *String* a ser utilizada como buffer de leitura deve ser sempre pré-alocado e passado como referência. Caso contrário, os dados não poderão ser retornados.

**Sintaxe: FREAD ( < nHandle > , < cBuffer > , < nQtdBytes > )**

**Parâmetros:**

<b>nHandle</b>	É o manipulador (Handle) retornado pelas funções FOPEN(), FCREATE(), FOPENPORT(), que faz referência ao arquivo a ser lido.
<b>cBuffer</b>	É o nome de uma variável do tipo <i>String</i> , a ser utilizada como <i>buffer</i> de leitura, onde os dados lidos deverão ser armazenados. O tamanho desta variável deve ser maior ou igual ao tamanho informado em nQtdBytes. Esta variável deve ser sempre passada por referência. (@ antes do nome da variável), caso contrário os dados lidos não serão retornados.

<b>nQtdBytes</b>	Define a quantidade de <i>Bytes</i> que devem ser lidas do arquivo a partir posicionamento do ponteiro atual.
------------------	---

**Retorno:**

<b>Numérico</b>	Quantidades de <i>bytes</i> lidos. Caso a quantidade seja menor que a solicitada, isto indica erro de leitura ou final de arquivo, Verifique a função FERROR() para maiores detalhes.
-----------------	---

## FREADSTR ()

---

Função que realiza a leitura de caracteres de um arquivo binário. FREADSTR() lê de um arquivo aberto, através de FOPEN(), FCREATE(), FOPENPORT(). FREADSTR() lerá até o número de bytes informado em nQtdBytes ou até encontrar um CHR(0). Caso aconteça algum erro ou o arquivo chegue ao final, FREADSTR() retornará uma *string* menor do que nQdBytes e colocará o erro em FERROR(). FREADSTR() lê a partir da posição atual do ponteiro, que pode ser ajustado pelo FSEEK(), FWRITE( ) ou FREAD().

**Sintaxe:** FREADSTR ( < nHandle > , < nQtdBytes > )

**Parâmetros:**

<b>nHandle</b>	É o manipulador retornado pelas funções FOPEN(), FCREATE(), FOPENPORT().
<b>nQtdBytes</b>	Número máximo de bytes que devem ser lidos.

**Retorno:**

<b>Caracter</b>	Retorna uma <i>string</i> contendo os caracteres lidos.
-----------------	---

## FRENAME()

---

Através da função FRENAME() é possível renomear um arquivo para outro nome, tanto no servidor como na estação. Ao renomear um arquivo não esqueça que esta arquivo deverá estar fechado (isto é , não pode estar em uso por nenhum outro processo ou estação). Caso o arquivo esteja aberto por outro processo, a operação de renomear o arquivo não é possível. A função fRename() não aceita *wildcards* ( \* e/ou ? ).

Vale lembrar que não é possível renomear um arquivo especificando nos parâmetros simultaneamente um caminho de servidor e um de estação remota, bem como especificar dois arquivos remotos e executar a função fRename() através de um JOB. Caso isto ocorra, a função retornará -1 , e fError() retornará também -1.



**Importante**

Quando especificamos um path diferente nos arquivos de origem e destino , a função fRename() realiza a funcionalidade de MOVER o arquivo para o Path especificado.

**Sintaxe: FRENAMEN ( < cOldFile > , < cNewFile > )**

**Parâmetros:**

<b>cOldFile</b>	Nome do arquivo será renomeado, aceita caminhos do servidor e caminhos do cliente. Caso não seja especificado nenhuma unidade de disco e pacote, é considerado o pacote atual no servidor.
<b>cNewFile</b>	Novo nome do arquivo, aceita também caminho do servidor, e caminho do cliente

**Retorno:**

<b>Numérico</b>	Se o <i>status</i> retornado for -1 , ocorreu algum erro na mudança de nome: Verifique se os dois caminhos estão no mesmo ambiente, verifique a existência do arquivo de origem, se ele não está em uso no momento por outro processo, e verifique se o nome do arquivo de destino já não existe no pacote de destino especificado.
-----------------	--

### FSEEK()

Função que posiciona o ponteiro do arquivo para as próximas operações de leitura ou gravação. As movimentações de ponteiros são relativas à nOrigem que pode ter os seguintes valores, definidos em fileio.ch:

**Tabela A: Origem a ser considerada para a movimentação do ponteiro de posicionamento do Arquivo.**

Origem	Constate(fileio.ch)	Operação
0	FS_SET	Ajusta a partir do inicio do arquivo. (Default)
1	FS_RELATIVE	Ajuste relativo a posição atual do arquivo.
2	FS_END	Ajuste a partir do final do arquivo.

**Sintaxe: FSEEK ( < nHandle > , [ nOffSet ] , [ nOrigem ] )**

**Parâmetros:**

<b>nHandle</b>	Manipulador obtido através das funções FCREATE,FOPEN.
<b>nOffSet</b>	<u>nOffSet corresponde ao número de bytes no ponteiro de posicionamento do arquivo a ser movido. Pode ser um número positivo, zero ou negativo, a ser considerado a partir do parâmetro passado em nOrigem.</u>
<b>nOrigem</b>	Indica a partir de qual posição do arquivo, o nOffset será considerado.

**Retorno:**

<b>Numérico</b>	FSEEK() retorna a nova posição do ponteiro de arquivo com relação ao início do arquivo (posição 0) na forma de um valor numérico inteiro. Este valor não leva em conta a posição original do ponteiro de arquivos antes da execução da função FSEEK().
-----------------	--

### **FT\_FEOF()**

Função que retorna verdadeiro (.t.) se o arquivo texto aberto pela função FT\_FUSE() estiver posicionado no final do arquivo, similar à função EOF() utilizada para arquivos de dados.

**Sintaxe: FT\_FEOF( )**

**Parâmetros:**

<b>Nenhum</b>	()
---------------	----

**Retorno:**

<b>Lógico</b>	Retorna true caso o ponteiro do arquivo tenha chegado ao final, false caso contrário.
---------------	---

### **FT\_FGOTO()**

Função utilizada para mover o ponteiro, que indica a leitura do arquivo texto, para a posição absoluta especificada pelo argumento <nPos>.

**Sintaxe: FT\_FGOTO ( < nPos > )**

**Parâmetros:**

<b>nPos</b>	Indica a posição em que será colocado o ponteiro para leitura dos dados no arquivo.
-------------	---

**Retorno:**

Nenhum	()
--------	----

---

**FT\_FGOTOP()**

---

A função tem como objetivo mover o ponteiro, que indica a leitura do arquivo texto, para a posição absoluta especificada pelo argumento <nPos>.

**Sintaxe: FT\_FGOTO ( < nPos > )**

**Parâmetros:**

nPos	Indica a posição que será colocado o ponteiro para leitura dos dados no arquivo.
------	--

**Retorno:**

Nenhum	()
--------	----

---

**FT\_FLASTREC()**

---

Função que retorna o número total de linhas do arquivo texto aberto pela FT\_FUse. As linhas são delimitadas pela seqüência de caracteres CRLF ou LF.



*Dica*

Verifique maiores informações sobre formato do arquivo e tamanho máximo da linha de texto na função FT\_FREADLN().

**Sintaxe: FT\_FLASTREC( )**

**Parâmetros:**

Nenhum	()
--------	----

**Retorno:**

Numérico	Retorna a quantidade de linhas existentes no arquivo. Caso o arquivo esteja vazio, ou não exista arquivo aberto, a função retornará 0 (zero).
----------	---

### Exemplo:

```
FT_FUse('teste.txt') // Abre o arquivo  
CONOUT("Linhas no arquivo ["+str(ft_flastrec(),6)+""]")  
FT_FGOTOP()  
While !FT_FEof()  
    conout("Ponteiro ["+str(FT_FRECNO(),6)+""] Linha  
["+FT_FReadln()+""]")  
    FT_FSkip()  
Enddo  
FT_FUse() // Fecha o arquivo
```

### FT\_FREADLN()

Função que retorna uma linha de texto do arquivo aberto pela FT\_FUse. As linhas são delimitadas pela seqüência de caracteres CRLF ( *chr(13) + chr(10)* ), ou apenas LF ( *chr(10)* ), e o tamanho máximo de cada linha é 1022 bytes.

- A utilização desta função não altera a posição do ponteiro para leitura dos dados, o ponteiro do arquivo não é movido. A movimentação do ponteiro é realizada através da função FT\_FSKIP().
- O limite de 1022 bytes por linha inclui os caracteres delimitadores de final de linha. Deste modo, quando utilizados os separadores CRLF, isto nos deixa 1020 bytes de texto, e utilizando LF, 1021 bytes. A tentativa de leitura de arquivos com linhas de texto maiores do que os valores especificados acima resultará na leitura dos 1023 primeiros bytes da linha, e incorreta identificação das quebras de linha posteriores.
- As funções FT\_F\* foram projetadas para ler arquivos com conteúdo texto apenas. A utilização das mesmas em arquivos binários pode gerar comportamentos inesperados na movimentação do ponteiro de leitura do arquivo, e incorretas identificações nos separadores de final de linha.



*Importante*



*Dica*

- **Release:** Quando utilizado um Protheus Server, com build superior a 7.00.050713P, a função FT\_FREADLN() também é capaz de ler arquivos texto / ASCII, que utilizam também o caractere LF ( *chr(10)* ) como separador de linha.

**Sintaxe: FT\_FREADLN( )**

**Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

**Retorno:**

<b>Caracter</b>
-----------------

	Retorna a linha inteira na qual está posicionado o ponteiro para leitura de dados.
--	--

---

**FT\_FRECNO()**

---

A função tem o objetivo de retornar a posição do ponteiro do arquivo texto.

A função FT\_FRecno retorna a posição corrente do ponteiro do arquivo texto aberto pela FT\_FUse.

**Sintaxe: FT\_FRECNO ( )**

**Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

**Retorno:**

<b>Caracter</b>
-----------------

	Retorna a posição corrente do ponteiro do arquivo texto.
--	--

---

**FT\_FSKIP()**

---

Função que move o ponteiro do arquivo texto aberto pela FT\_FUSE() para a próxima linha, similar ao DBSKIP() usado para arquivos de dados.

**Sintaxe: FT\_FSKIP ( [ nLinhas ] )**

**Parâmetros:**

<b>nLinhas</b>
----------------

	nLinhas corresponde ao número de linhas do arquivo TXT ref. movimentação do ponteiro de leitura do arquivo.
--	---

**Retorno**

<b>Nenhum</b>
---------------

( )
-----

## **FT\_FUSE()**

---

Função que abre ou fecha um arquivo texto para uso das funções FT\_F\*. As funções FT\_F\* são usadas para ler arquivos texto, onde as linhas são delimitadas pela seqüência de caracteres CRLF ou LF (\*) e o tamanho máximo de cada linha é 1022 *bytes*. O arquivo é aberto em uma área de trabalho, similar à usada pelas tabelas de dados.

---



Dica

Verifique maiores informações sobre formato do arquivo e tamanho máximo da linha de texto na função FT\_FREADLN().

---

**Sintaxe:** **FT\_FUSE ( [ cTXTFile ] )**

**Parâmetros:**

<b>cTXTFile</b>	Corresponde ao nome do arquivo TXT a ser aberto. Caso o nome não seja passado, e já exista um arquivo aberto. o mesmo é fechado.
-----------------	--

**Retorno:**

<b>Numérico</b>	A função retorna o Handle de controle do arquivo. Em caso de falha de abertura, a função retornará -1
-----------------	---

## **FWRITE()**

---

Função que permite a escrita em todo ou em parte do conteúdo do *buffer*, limitando a quantidade de Bytes através do parâmetro nQtdBytes. A escrita começa a partir da posição corrente do ponteiro de arquivos, e a função FWRITE retornará a quantidade real de bytes escritos. Através das funções FOPEN(), FCREATE(), ou FOPENPORT(), podemos abrir ou criar um arquivo ou abrir uma porta de comunicação, para o qual serão gravados ou enviados os dados do buffer informado. Por tratar-se de uma função de manipulação de conteúdo binário , são suportados na String cBuffer todos os caracteres da tabela ASCII , inclusive caracteres de controle ( ASC 0 , ASC 12 , ASC 128 , etc.).

Caso aconteça alguma falha na gravação, a função retornará um número menor que o nQtdBytes. Neste caso , a função FERROR() pode ser utilizada para determinar o erro específico ocorrido. A gravação no arquivo é realizada a partir da posição atual do ponteiro, que pode ser ajustado através das funções FSEEK(), FREAD() ou FREADSTR().

**Sintaxe:** **FWRITE ( < nHandle > , < cBuffer > , [ nQtdBytes ] )**

**Parâmetros:**

<b>nHandle</b>	É o manipulador de arquivo ou <i>device</i> retornado pelas funções FOPEN(), FCREATE(), ou FOPENPORT().
<b>cBuffer</b>	<cBuffer> é a cadeia de caracteres a ser escrita no arquivo especificado. O tamanho desta variável deve ser maior ou igual ao tamanho informado em nQtdBytes (caso seja informado o tamanho).
<b>nQtdBytes</b>	<nQtdBytes> indica a quantidade de bytes a serem escritos a partir da posição corrente do ponteiro de arquivos. Caso seja omitido, todo o conteúdo de <cBuffer> é escrito.

**Retorno:**

<b>Numérico</b>	FWRITE() retorna a quantidade de bytes escritos na forma de um valor numérico inteiro. Caso o valor retornado seja igual a <nQtdBytes>, a operação foi bem sucedida. Caso o valor de retorno seja menor que <nBytes> ou zero, ou o disco está cheio ou ocorreu outro erro. Neste caso, utilize a função FERROR() para obter maiores detalhes da ocorrência.
-----------------	---

**Exemplo:**

```
#INCLUDE "FILEIO.CH"
#define F_BLOCK 1024 // Define o bloco de Bytes a serem lidos /
gravados por vez

User Function TestCopy()
Local cBuffer := SPACE(F_BLOCK)
Local nHOrigem , nHDestino
Local nBytesLidos , nBytesFalta , nTamArquivo
Local nBytesLer , nBytesSalvo
Local ICopiaOk := .T.

// Abre o arquivo de Origem
nHOrigem := FOPEN("ORIGEM.TXT", FO_READ)

// Testa a abertura do Arquivo
If nHOrigem == -1
    MsgStop('Erro ao abrir origem. Ferror =
'+str(ferror(),4),'Erro')
    Return .F.
Endif

// Determina o tamanho do arquivo de origem
nTamArquivo := Fseek(nHOrigem,0,2)

// Move o ponteiro do arquivo de origem para o inicio do arquivo
Fseek(nHOrigem,0)

// Cria o arquivo de destino
nHDestino := FCREATE("DESTINO.TXT", FC_NORMAL)

// Testa a criação do arquivo de destino
If nHDestino == -1
    MsgStop('Erro ao criar destino. Ferror =
'+str(ferror(),4),'Erro')
    FCLOSE(nHOrigem) // Fecha o arquivo de Origem
```

```

        Return .F.
Endif

// Define que a quantidade que falta copiar é o próprio tamanho do
Arquivo
nBytesFalta := nTamArquivo

// Enquanto houver dados a serem copiados
While nBytesFalta > 0

    // Determina quantidade de dados a serem lidos
    nBytesLer := Min(nBytesFalta , F_BLOCK )

    // Lê os dados do Arquivo
    nBytesLidos := FREAD(nHOrigem, @cBuffer, nBytesLer )

    // Determina se não houve falha na leitura
    If nBytesLidos < nBytesLer
        MsgStop(      "Erro de Leitura da Origem. "+;
                      Str(nBytesLer,8,2)+" bytes a LER."+;
                      Str(nBytesLidos,8,2)+" bytes Lidos."+;
                      "Ferror = "+str(ferror(),4),'Erro')
        ICopiaOk := .F.
        Exit
    Endif

    // Salva os dados lidos no arquivo de destino
    nBytesSalvo := FWRITE(nHDestino, cBuffer,nBytesLer)

    // Determina se não houve falha na gravação
    If nBytesSalvo < nBytesLer
        MsgStop("Erro de gravação do Destino. "+;
                  Str(nBytesLer,8,2)+" bytes a SALVAR."+;
                  Str(nBytesSalvo,8,2)+" bytes
gravados."+;
                  "Ferror = "+str(ferror(),4),'Erro')
        ICopiaOk := .F.
        EXIT
    Endif

    // Elimina do Total do Arquivo a quantidade de bytes copiados
    nBytesFalta -= nBytesLer

Enddo

// Fecha os arquivos de origem e destino
FCLOSE(nHOrigem)
FCLOSE(nHDestino)

If ICopiaOk
    MsgStop('Cópia de Arquivos finalizada com sucesso. '+;
            str(nTamArquivo,12,0)+' bytes
copiados.','Final')
Else
    MsgStop( 'Falha na Cópia. Arquivo de Destino incompleto. '+;
              'Do total de '+str(nTamArquivo,12,0)+' bytes,
faltaram '+str(nBytesFalta,12,0)+' bytes.','Final')
Endif

Return

```

## **MSCOPYFILE()**

---

Função que executa a cópia binária de um arquivo para o destino especificado.

**Sintaxe:** **MSCOPYFILE( cArqOrig, cArqDest )**

**Parâmetros:**

<b>cArqOrig</b>	Nome do arquivo origem e a extensão.
<b>cArqDest</b>	Nome do arquivo destino e a extensão.

**Retorno:**

<b>Lógico</b>	Se a copia for realizada com sucesso a função retornará verdadeiro (.T.).
---------------	---

**Exemplo:**

```
Local cArqOrig := 'ARQ00001.DBF'  
Local cArqDest := 'ARQ00002.XXX'  
  
If MsCopyFile( cArqOrig, cArqDest )  
    APMsgInfo('Copia realizada com sucesso!')  
EndIf
```

---

## **MSCOPYTO()**

---

Função que realiza a cópia dos registros de uma base de dados para outra, criando o arquivo destino de acordo com a estrutura da base de dados origem.

**Sintaxe:** **MSCOPYTO( [cArqOrig], cArqDest )**

**Parâmetros:**

<b>cArqOrig</b>	Nome do arquivo origem e a extensão se o ambiente for Top o parâmetro passará a ser obrigatório.
<b>cArqDest</b>	Nome do arquivo destino e a extensão.

**Retorno:**

<b>Lógico</b>	Se a cópia for realizada com sucesso a função retornará verdadeiro (.T.).
---------------	---

**Exemplo:**

```
Local cArqDest := 'SX2ZZZ.DBF'
```

```

DbSelectArea('SX2')
If MsCopyTo( , cArqDest )
APMsgInfo('Copia realizada com sucesso!')
Else
APMsgInfo('Problemas ao copiar o arquivo SX2!')
EndIf

```

## **MSCREATE()**

---

Função que cria um arquivo (tabela) de acordo com a estrutura informada no parâmetro aStruct. Se o parâmetro cDriver não for informado o RDD corrente será assumido como padrão. Para criação de tabelas no Top Connect é necessário estar conectado ao banco e o environment do protheus ser TOP.

---



*Importante*

aStruct: *array* contendo a estrutura da tabela aonde:

- 1º - caracter, nome do campo;
- 2º - caracter, tipo do campo;
- 3º - numérico, tamanho do campo;
- 4º - numérico, decimais.

**Sintaxe: MsCreate( cArquivo, aStrut ,[cDriver] )**

**Parâmetros:**

<b>cArquivo</b>	Nome do arquivo.
<b>aStruct</b>	Estrutura do arquivo.
<b>cDriver</b>	RDD do arquivo.

**Retorno:**

<b>Lógico</b>	Indica se a operação foi executada com sucesso.
---------------	---

**Exemplo:**

```

Local cTarget := '\sigaadv\' 
Local aStrut
aStrut := { { 'Campo', 'C', 40, 0 } }
If MsCreate( cTarget+'ARQ1001', aStrut )
APMsgInfo('Criado com sucesso!')
Else
APMsgInfo('Problemas ao criar o arquivo!')
EndIf

```

## **MSERASE()**

---

Função utilizada para deletar fisicamente o arquivo especificado.

**Sintaxe:** **MsErase( cArquivo, [cIndice], [cDriver] )**

**Parâmetros:**

<b>cArquivo</b>	Nome do arquivo e a extensão.
<b>cIndice</b>	Nome do arquivo de índice e a extensão.
<b>cDriver</b>	RDD do arquivo, se não for informado assumirá o RDD corrente.

**Retorno:**

<b>Lógico</b>	Indica se a operação foi executada com sucesso.
---------------	---

**Exemplo:**

```
Local cArquivo := 'SX2ZZZ.DBF'  
Local cIndice := 'SX2ZZZ'+ OrdBagExt()  
If MsErase( cArquivo, cIndice )  
  APMsgInfo( 'Arquivo deletado com sucesso!' )  
Else  
  APMsgInfo( 'Problemas ao deletar arquivo!' )  
EndIf
```

## **MSRENAME()**

---

Função que verifica a existência do arquivo especificado.

**Sintaxe:** **MsFile( cArquivo, [cIndice], [cDriver] )**

**Parâmetros:**

<b>cArquivo</b>	Nome do arquivo e a extensão.
<b>cIndice</b>	Nome do arquivo de índice e a extensão.
<b>cDriver</b>	RDD do arquivo, se não for informado assumirá o RDD corrente.

**Retorno:**

<b>Lógico</b>	Indica se o arquivo especificado existe.
---------------	--

**Exemplo:**

```
Local cArquivo := 'SX2ZZZ.DBF'  
Local clndice := 'SX2ZZZ'+ OrdBagExt()  
If !MsFile ( cArquivo, clndice )  
APMsgInfo( 'Arquivo não encontrado!' )  
EndIf
```

**RETFILENAME()**

---

Função que retorna o nome de um arquivo contido em uma *string*, ignorando o caminho e a extensão.

**Sintaxe:** RetFileName( cArquivo )

**Parâmetros:**

cArquivo	<i>String</i> contendo o nome do arquivo
----------	--

**Retorno:**

Caracter	Nome do arquivo contido na <i>string</i> cArquivo sem o caminho e a extensão.
----------	---

**Exemplo:**

```
Local cArquivo := '\SIGAADV\ARQZZZ.DBF'  
cArquivo := RetFileName( cArquivo )  
// retorno 'ARQZZZ'
```

**Manipulação de arquivos e índices temporários**

**CRIATRAB()**

Função que cria um arquivo de trabalho com uma estrutura especificada, sendo que:

- ☒ Caso o parâmetro IDbf seja definido como .T., a função criará um arquivo DBF com este nome e a estrutura definida em aArray.
- ☒ Caso o parâmetro IDbf seja definido como .F., a função não criará arquivo de nenhum tipo, apenas fornecerá um nome válido.

**Sintaxe:** CriaTrab(aArray, IDbf)

**Parâmetros:**

<b>aArray</b>	Array multidimensional contendo a estrutura de campos da tabela que será criada no formato: {Nome, Tipo, Tamanho, Decimal}
<b>IDbf</b>	Determina se o arquivo de trabalho deve ser criado ( .T.) ou não (.F. )

**Retorno:**

<b>Caracter</b>	Nome do Arquivo gerado pela função.
-----------------	-------------------------------------

**Exemplo:**

```
// Com IDbf = .F.  
cArq := CriaTrab(NIL, .F.)  
cIndice := "C9_AGREG+"+IndexKey()  
Index on &cIndice To &cArq  
  
// Com IDbf = .T.  
aStru := {}  
AADD(aStru, { "MARK", "C", 1, 0})  
AADD(aStru, { "AGLUT", "C", 10, 0})  
AADD(aStru, { "NUMOP", "C", 10, 0})  
AADD(aStru, { "PRODUTO", "C", 15, 0})  
AADD(aStru, { "QUANT", "N", 16, 4})  
AADD(aStru, { "ENTREGA", "D", 8, 0})  
AADD(aStru, { "ENTRAJU", "D", 8, 0})  
AADD(aStru, { "ORDEM", "N", 4, 0})  
AADD(aStru, { "GERADO", "C", 1, 0})  
cArqTrab := CriaTrab(aStru, .T.)  
USE &cArqTrab ALIAS TRB NEW
```



*Importante*

Na criação de índices de trabalho temporários é utilizada a sintaxe:

**CriaTrab(Nil, .F.)**

## Manipulação de bases de dados

### **ALIAS()**

Função de banco de dados utilizada para determinar o alias da área de trabalho especificada. Alias é o nome atribuído a uma área de trabalho quando um arquivo de banco de dados está em uso. O nome real atribuído é o nome do arquivo de banco de dados, ou um nome que foi explicitamente atribuído através da cláusula ALIAS do comando USE.

A função ALIAS() é o inverso da função SELECT() pois retorna o alias através do número da área de trabalho, enquanto SELECT() retorna o número da área de trabalho através do alias.

**Sintaxe:** ALIAS ( [ nAreaTrabalho ] )

**Parâmetros:**

<b>nAreaTrabalho</b>	<nAreaTrabalho> é o número da área de trabalho a ser verificada.
----------------------	--

**Retorno:**

<b>Caracter</b>	Retorna o alias da área de trabalho especificada na forma de uma cadeia de caracteres, em letra maiúscula. Caso a <nAreaTrabalho> não seja especificada, é retornado o alias da área de trabalho corrente. Se não houver nenhum arquivo de banco de dados em uso na Área de Trabalho especificada, ALIAS() retorna uma cadeia de caracteres nula ("").
-----------------	--

**Exemplo:**

```
cAlias := alias()
IF empty(cAlias)
    alert('Não há Área em uso')
Else
    alert(Area em uso atual : '+cAlias)
Endif
```

### **BOF() / EOF()**

As funções BOF() e EOF() são utilizadas para determinar se o ponteiro de leitura do arquivo encontra-se no começo ou no final do mesmo conforme abaixo:

- ☒ BOF() é uma função de tratamento de banco de dados utilizada para testar uma condição de limite de inicial do arquivo quando o ponteiro de registros está se movendo para trás em um arquivo de banco de dados.

- EOF() é uma função de tratamento de banco de dados utilizada para testar uma condição de limite de final de arquivo quando o ponteiro de registros está se movendo para frente em um arquivo de banco de dados.

Normalmente é utilizada a condição EOF() como parte do argumento <Condicao> de uma construção DO WHILE que processa registros sequencialmente em um arquivo de banco de dados. Neste caso <Condicao> incluiria um teste para .NOT. EOF(), forçando o laço DO WHILE a terminar quando EOF() retornar verdadeiro (.T.)

#### Sintaxe: BOF() / EOF()

#### Parâmetros:

Nenhum	()
--------	----

#### Retorno:

Lógico	Retorna verdadeiro (.T.) quando feita uma tentativa de mover o ponteiro de registros para além do primeiro registro lógico em um arquivo de banco de dados. Do contrário, ela retorna falso (.F.).
Lógico	Retorna verdadeiro (.T.) quando feita uma tentativa de mover o ponteiro de registros para além do último registro lógico em um arquivo de banco de dados. Do contrário, ela retorna falso (.F.). Caso não haja nenhum arquivo de banco de dados aberto na área de trabalho corrente, EOF() retorna falso (.F.). Se o arquivo de banco de dados corrente não possui registros, EOF() retorna verdadeiro (.T.).



## **COPY()**

---

O comando COPY TO permite a cópia de todos ou parte dos registros da tabela atualmente selecionada como área de trabalho atual, para um novo arquivo. Os registros considerados para a cópia podem ser limitados pela cláusula <escopo>, através de expressões FOR/WILE, e/ou através de um filtro.

Se o filtro para registros deletados ( SET DELETED ) estiver desligado (OFF), registros deletados ( marcados para deleção ) são copiados para o arquivo de destino, mantendo este *status*. Caso contrário, nenhum registro deletado é copiado. Da mesma maneira, caso exista uma condição de filtro na tabela atual ( SET FILTER ), apenas os registros que satisfaçam a condição de filtro serão copiados.

Os registros são lidos na tabela atual, respeitando a ordem de índice setada. Caso não hajam índices abertos, ou a ordem de navegação nos índices (SET ORDER ) seja 0 (zero), os registros são lidos em ordem natural ( ordem de RECNO ).

A tabela de destino dos dados copiados é criada, e aberta em modo exclusivo, antes da operação de cópia efetiva ser iniciada.

**Tabela A : Especificação do formato SDF ( System Data Format )**

<b>Elemento do Arquivo</b>	<b>Formato</b>
<b>Campos 'C' Caractere</b>	Tamanho fixo, ajustado com espaços em branco.
<b>Campos 'D' Data</b>	Formato aaaammdd ( ano, mês, dia ).
<b>Campos 'L' lógicos</b>	T ou F.
<b>Campos 'M' Memo</b>	(campo ignorado).
<b>Campos 'N' Numéricos</b>	Ajustados à direita, com espaços em branco.
<b>Delimitador de Campos</b>	Nenhum.
<b>Separador de Registros</b>	CRLF ( ASCII 13 + ASCII 10 ).
<b>Marca de final de arquivo (EOF)</b>	Nenhum.

**Tabela B : Especificação do formato delimitado ( DELIMITED / DELIMITED WITH <cDelimiter> )**

Elemento do Arquivo	Formato
Campos 'C' Caractere	Delimitados, ignorando espaços à direita.
Campos 'D' Data	Formato aaaammdd ( ano, mês, dia ).
Campos 'L' lógicos	T ou F.
Campos 'M' Memo	(campo ignorado).
Campos 'N' Numéricos	sem espaços em branco.
Delimitador de Campos	Vírgula.
Separador de Registros	CRLF ( ASCII 13 + ASCII 10 ).
Marca de final de arquivo (EOF)	Nenhum.



*Importante*

A Linguagem Advpl, antes do Protheus, suportava a geração de uma tabela delimitada diferenciada, obtida através do comando *COPY TO (...) DELIMITED WITH BLANK*. No Protheus este formato não é suportado. Caso utilize-se deste comando com a sintaxe acima, o arquivo ASCII gerado será delimitado, utilizando-se a sequência de caracteres 'BLANK' como delimitadora de campos Caractere.

#### Sintaxe:

```
COPY [ FIELDS <campo,...> ] TO cFile [cEscopo] [ WHILE <lCondicao> ]
[ FOR <lCondicao> ] [ SDF | DELIMITED [WITH <cDelimiter>] ]
[ VIA <cDriver> ]
```

#### Parâmetros:

<b>FIELDS &lt;campo,...&gt;</b>	FIELDS <campo,...> especifica um ou mais campos, separados por vírgula, a serem copiados para a tabela de destino. Caso não especificado este parâmetro, serão copiados todos os campos da tabela de origem.
<b>TO cFile</b>	TO <cFile> especifica o nome do arquivo de destino. O nome do arquivo de destino pode ser especificado de forma literal direta, ou como uma expressão Advpl, entre parênteses.  Caso sejam especificadas as cláusulas SDF ou DELIMITED, é gerado um arquivo ASCII, com extensão .txt por default.
<b>cEscopo</b>	<cEscopo> define a porção de dados da tabela atual a ser copiada. Por default, são copiados todos os registros (ALL). Os escopos possíveis de uso são:

	<p>ALL - Copia todos os registros.</p> <p>REST - Copia, a partir do registro atualmente posicionado, até o final da tabela.</p> <p>NEXT &lt;n&gt; - Copia apenas &lt;n&gt; registros, iniciando a partir do registro atualmente posicionado.</p> <p><b>OBSERVAÇÃO :</b> Vale a pena lembrar que o escopo é sensível também às demais condições de filtro ( WHILE / FOR ).</p>
<b>WHILE &lt;lCondicao&gt;</b>	WHILE <lCondicao> permite especificar uma condição para realização da cópia, a partir do registro atual, executada antes de inserir cada registro na tabela de destino, sendo realizada a operação de cópia enquanto esta condição for verdadeira.
<b>FOR &lt;lCondicao&gt;</b>	FOR <lCondicao> especifica uma condição para cópia de registros, executada antes de inserir um registro na tabela de destino, sendo a operação realizada apenas se lCondicao for verdadeira ( .T. )
<b>[SDF   DELIMITED]</b>	<p>[ SDF   DELIMITED [WITH &lt;xcDelimiter&gt;] ]</p> <p>SDF especifica que o tipo de arquivo de destino gerado é um arquivo no formato "System Data Format" ASCII, onde registros e campos possuirem tamanho fixo no arquivo de destino.</p> <p>DELIMITED especifica que o arquivo ASCII de destino será no formato delimitado, onde os campos do tipo Caractere são delimitados entre aspas duplas ( delimitador Default ). Registros e campos têm tamanho variável no arquivo ASCII.</p> <p>DELIMITED WITH &lt;xcDelimiter&gt; permite especificar um novo caractere, ou sequência de caracteres, a ser utilizada como delimitador, ao invés do default ( aspas duplas ). O caractere delimitador pode ser escrito de forma literal, ou como uma expressão entre parênteses.</p> <p>Nas Tabelas complementares A e B, na documentação do comando, são detalhadas as especificações dos formatos SDF e DELIMITED.</p>
<b>VIA &lt;cDriver&gt;</b>	<p>VIA &lt;xcDriver&gt; permite especificar o driver utilizado para criar a tabela de destino dos dados a serem copiados.</p> <p>O Driver deve ser especificado como uma expressão caractere. Caso especificado como um valor literal direto, o mesmo deve estar entre aspas.</p>

**Retorno:**

Nenhum	()
--------	----

**COPY STRUCTURE()**

---

O comando COPY STRUCTURE cria uma nova tabela vazia, com a estrutura da tabela ativa na área de trabalho atual. Caso a tabela a ser criada já exista, a mesma é sobreescrita. A tabela de destino criada utiliza o mesmo RDD da tabela de origem ( tabela ativa na área de trabalho atual ).



**Importante**

A Linguagem Advpl, antes do Protheus, suportava a parametrização de uma lista de campos da tabela ativa, para compor a estrutura da tabela de destino, através da cláusula `FIELDS <campo,...>`. Esta opção não é suportada no Protheus. Caso seja utilizada, o programa será abortado com a ocorrência de erro fatal : '**DBCopyStruct - Parameter <Fields> not supported in Protheus'**

---

**Sintaxe:**

**COPY STRUCTURE TO <xcDataBase>**

**Parâmetros:**

<b>TO &lt;xcDataBase&gt;</b>	Deve ser especificado em xcDatabase o nome da tabela a ser criada.
------------------------------	--

**Retorno:**

Nenhum	()
--------	----

**DBAPPEND()**

---

A função DBAPPEND() acrescenta mais um registro em branco no final da tabela corrente. Se não houver erro da RDD, o registro é acrescentado e bloqueado.

**Sintaxe:** DBAPPEND ( [ ILiberaBloqueios ] )

**Parâmetros:**

ILiberaBloqueios	Se o valor for .T., libera todos os registros bloqueados anteriormente (locks). Se for .F., todos os bloqueios anteriores são mantidos. Valor default: .T.
------------------	---

**Retorno:**

Nenhum	()
--------	----

**Exemplo:**

```
USE Clientes NEW
FOR i:=1 to 5
    DBAPPEND(.F.)
    NOME := "XXX"
    END := "YYY"
NEXT
// Os 5 registros incluídos permanecem bloqueados
DBAPPEND()
// Todos os bloqueios anteriores são liberados
```

**DBCLEARALLFILTER()**

A função DBCLEARALLFILTER() salva as atualizações realizadas e pendentes de todas as tabelas e depois limpa as condições de filtro de todas as tabelas.

**Sintaxe: DBCLEARALLFILTER()**

**Parâmetros:**

Nenhum	()
--------	----

**Retorno:**

Nenhum	()
--------	----

**Exemplo:**

```
USE Clientes NEW
DBSETFILTER( {|| Idade < 40}, 'Idade < 40') // Seta a expressão de
filtro
...
DBCLEARALLFILTER()
// Limpa a expressão de filtro de todas as ordens
```

## **DBCLEARFILTER()**

---

A função DBCLEARFILTER() salva as atualizações realizadas e pendentes na tabela corrente e depois limpa todas as condições de filtro da ordem ativa no momento. Seu funcionamento é oposto ao comando SET FILTER.

### **Sintaxe: DBCLEARFILTER()**

#### **Parâmetros:**

<b>Nenhum</b>	<b>()</b>
---------------	-----------

#### **Retorno:**

<b>Nenhum</b>	<b>()</b>
---------------	-----------

#### **Exemplo:**

```
USE Clientes NEW
DBSETFILTER( {|| Idade < 40}, "Idade < 40" ) // Seta a expressão de
filtro
...
DBCLEARFILTER()
// Limpa a expressão de filtro
```

## **DBCLEARINDEX()**

---

A função DBCLEARINDEX() salva as atualizações pendentes na tabela corrente e fecha todos os arquivos de índice da área de trabalho. Por consequência, limpa todas as ordens da lista. Seu funcionamento é oposto ao comando SET INDEX.

### **Sintaxe: DBCLEARINDEX()**

#### **Parâmetros:**

<b>Nenhum</b>	<b>()</b>
---------------	-----------

#### **Retorno:**

<b>Nenhum</b>	<b>()</b>
---------------	-----------

#### **Exemplo:**

```
USE Clientes NEW
DBSETINDEX("Nome") // Abre o arquivo de índice "Nome"
...
```

```
DBCLEARINDEX()  
// Fecha todos os arquivos de índices
```

## **DBCLOSEALL()**

---

A função DBCLOSEALL() salva as atualizações pendentes, libera todos os registros bloqueados e fecha todas as tabelas abertas (áreas de trabalho) como se chamassem DBCLOSEAREA para cada área de trabalho.

**Sintaxe: DBCLOSEALL()**

**Parâmetros:**

<b>Nenhum</b>	()
---------------	----

**Retorno:**

<b>Nenhum</b>	()
---------------	----

**Exemplo:**

```
// Este exemplo demonstra como se pode utilizar o DBCLOSEALL para  
fechar a área de trabalho atual.  
USE Clientes NEW  
DBSETINDEX("Nome") // Abre o arquivo de índice "Nome"  
USE Fornecedores NEW  
DBSETINDEX("Idade") // Abre o arquivo de índice "Idade"  
...  
DBCLOSEALL() //Fecha todas as áreas de trabalho, todos os índices e  
ordens
```

## **DBCLOSEAREA()**

---

A função DBCLOSEAREA() permite que um alias presente na conexão seja fechado, o que viabiliza seu reuso em outra operação. Este comando tem efeito apenas no alias ativo na conexão, sendo necessária sua utilização em conjunto com o comando DbSelectArea().

**Sintaxe: DBCLOSEAREA()**

**Parâmetros:**

<b>Nenhum</b>	()
---------------	----

**Retorno:**

Nenhum	()
--------	----

**Exemplo:**

```
DbUserArea(.T., "DBFCDX", "\SA1010.DBF", "SA1DBF", .T., .F.)  
DbSelectArea("SA1DBF")  
MsgInfo("A tabela SA1010.DBF possui:" + STRZERO(RecCount(),6) + "  
registros.")  
DbCloseArea()
```

---

## DBCOMMIT()

A função DBCOMMIT() salva em disco todas as atualizações pendentes na área de trabalho corrente.

**Sintaxe: DBCOMMIT()**

**Parâmetros:**

Nenhum	()
--------	----

**Retorno:**

Nenhum	()
--------	----

**Exemplo:**

```
USE Clientes NEW  
DBGOTO(100)  
Nome := "Jose"  
USE Fornecedores NEW  
DBGOTO(168)  
Nome := "Joao"  
DBCOMMIT() // Salva em disco apenas as alterações realizadas na  
tabela Fornecedores
```

---

## DBCOMMITALL()

A função DBCOMMITALL() salva em disco todas as atualizações pendentes em todas as áreas de trabalho.

**Sintaxe: DBCOMMITALL()**

**Parâmetros:**

Nenhum	()
--------	----

**Retorno:**

Nenhum	()
--------	----

**Exemplo:**

```
USE Clientes NEW
DBGOTO(100)
Nome := "Jose"
USE Fornecedores NEW
DBGOTO(168)
Nome := "Joao"
DBCOMMITALL()
// Salva em disco as alterações realizadas nas tabelas Clientes e
Fornecedores
```

**DBCREATE()**

A função DBCREATE() é utilizada para criar um novo arquivo de tabela cujo nome está especificado através do primeiro parâmetro (cNome) e estrutura através do segundo (aEstrutura). A estrutura é especificada através de um *array* com todos os campos, onde cada campo é expresso através de um *array* contendo {Nome, Tipo, Tamanho, Decimais}.

**Sintaxe:** DBCREATE ( < cNOME > , < aESTRUTURA > , [ cDRIVER ] )

**Parâmetros:**

<b>cNOME</b>	Nome do arquivo a ser criado. Se contém pasta, ela se localiza abaixo do "RootPath". Se não, é criado por padrão no caminho formado por "RootPath"+"StartPath"
<b>aESTRUTURA</b>	Lista com as informações dos campos para ser criada a tabela.
<b>cDRIVER</b>	<u>Nome da RDD a ser utilizado para a criação da tabela. Se for omitido será criada com a corrente.</u>

**Retorno:**

Nenhum	()
--------	----

**Exemplo:**

```
// Este exemplo mostra como se pode criar novo arquivo através da
função DBCREATE:
LOCAL aEstrutura :={{Cod,N,3,0},
                     {Nome,C,10,0},
                     {Idade,N,3,0},
                     {Nasc,D,8,0},
                     {Pagto,N,7,2}}
// Cria a tabela com o RDD corrente
DBCREATE('\\teste\\cliente.dbf', aEstrutura)
USE '\\teste\\cliente.dbf' VIA 'DBFCDX' NEW
```

**Erros mais comuns:**

1. *DBCreate - Data base files can only be created on the server:* O nome do arquivo a ser criado não pode conter 'driver', pois, por convenção, ele seria criado na máquina onde o Remote está rodando.
2. *DBCreate - Invalid empty filename:* Nome do arquivo não foi especificado.
3. *DBCreate - Field's name cannot be 'DATA':* Algumas RDD's não suportam este nome de campo. É uma palavra reservada.
4. *DBCreate - The length of Field's name must be at most 10:* Nome do campo não pode ter mais que 10 caracteres.
5. *DBCreate - Field's name must be defined:* Nome do campo não foi definido.
6. *DBCreate - Field's type is not defined:* Tipo do campo não foi definido.
7. *DBCreate - invalid Field's type:* Tipo do campo é diferente de 'C', 'N', 'D', 'M' ou 'L'.

8. *DBCreate - Invalid numeric field format:* Considerando 'len' o tamanho total do campo numérico e 'dec' o número de decimais, ocorre este erro se:

- (len = 1) .and. (dec <> 0): Se o tamanho total é 1, o campo não pode ter decimais.
- (len>1) .and. (len< dec + 2): Se o tamanho total é maior que 1, ele deve ser maior que o número de decimais mais 2, para comportar o separador de decimais e ter pelo menos um algarismo na parte inteira.



*Importante*

**Exemplo:** O número 12.45 poderia ser o valor de um campo com len=5 e dec=2 (no mínimo).

---

### **Erros mais comuns:**



Podem ocorrer também erros decorrentes de permissão e direitos na pasta onde se está tentando criar o arquivo ou por algum problema no banco de dados. Verifique as mensagens do servidor Protheus e do banco de dados.

---

## **DBCREATEINDEX()**

---

A função DBCREATEINDEX() é utilizada para criar um novo arquivo de índice com o nome especificado através do primeiro parâmetro, sendo que, se o mesmo existir, é deletado e criado o novo. Para tanto são executados os passos a seguir:

- ❑ Salva fisicamente as alterações ocorridas na tabela corrente;
- ❑ Fecha todos os arquivos de índice abertos;
- ❑ Cria o novo índice;
- ❑ Seta o novo índice como a ordem corrente;
- ❑ Posiciona a tabela corrente no primeiro registro do índice.

Com exceção do RDD CTREE, a tabela corrente não precisa estar aberta em modo exclusivo para a criação de índice, pois na criação de índices no Ctree é alterada a estrutura da tabela, precisando para isto a tabela estar aberta em modo exclusivo.

**Sintaxe:**    **DBCREATEINDEX(<cNOME>, <cEXPCHAVE>, [bEXPCHAVE], [IUNICO])**

**Parâmetros:**

<b>cNOME</b>	Nome do arquivo de índice a ser criado.
<b>cEXPCHAVE</b>	Expressão das chaves do índice a ser criado na forma de <i>string</i> .
<b>bEXPCHAVE</b>	Expressão das chaves do índice a ser criado na forma executável.
<b>IUNICO</b>	Cria índice como único (o padrão é .F.).

**Retorno:**

<b>Nenhum</b>	()
---------------	----

**Exemplo:**

```
// Este exemplo mostra como se pode criar novo arquivo de índice  
criando a ordem sobre os  
// campos Nome e End e não aceitará duplicação:  
  
USE Cliente VIA "DBFCDX" NEW  
DBCREATEINDEX("\teste\ind2.cdx","Nome+End",{ || Nome+End },.T.)
```

**DBDELETE()**

---

A função DBDELETE() marca o registro corrente como “apagado” logicamente(), sendo necessária sua utilização em conjunto com as funções RecLock() e MsUnLock().

Para filtrar os registro marcados do alias corrente pode-se utilizar o comando SET DELETED e para apagá-los fisicamente pode-se utilizar a função \_\_DBPACK().

**Sintaxe:** DBDELETE ( )

**Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

**Retorno:**

<b>Nenhum</b>	( )
---------------	-----

**Exemplo:**

```
DbSelectArea("SA1")  
DbSetOrder(1) // A1_FILIAL + A1_COD + A1_LOJA  
DbSeek("01" + "900001" + "01") // Busca exata  
  
IF Found()  
    RecLock("SA1",.F.) // Define que será realizada uma alteração no  
    registro posicionado  
    DbDelete() // Efetua a exclusão lógica do registro posicionado.  
    MsUnLock() // Confirma e finaliza a operação  
ENDIF
```

## **DBF()**

---

A função DBF() verifica qual é o Alias da área de trabalho corrente. O Alias é definido quando a tabela é aberta através do parâmetro correspondente (DBUSEAREA()). Esta função é o inverso da função SELECT(), pois nesta é retornado o número da área de trabalho do Alias correspondente.

**Sintaxe:** DBF()

**Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

**Retorno:**

<b>Caracter</b>	Retorna o Alias corrente. Caso não exista Alias corrente retorna "" (String vazia).
-----------------	---

**Exemplo:**

```
dbUseArea( .T.,"dbfcxdads", "\dadosadv609\sa1990.dbf","SSS",.T., .F.  
)  
MessageBox("O Alias corrente é: "+DBF(),"Alias", 0) //Resultado: "O  
Alias corrente é: SSS"
```

---

## **DBFIELDINFO()**

A função DBFIELDINFO() é utilizada para obter informações sobre determinado campo da tabela corrente. O tipo de informação (primeiro argumento) é escolhido de acordo com as constantes abaixo:

**Tabela A : Constantes utilizadas na parametrização da função**

Constante	Descrição	Retorno
<b>DBS_NAME</b>	Nome do campo.	Caracter
<b>DBS_DEC</b>	Número de casas decimais.	Numérico
<b>DBS_LEN</b>	Tamanho.	Numérico
<b>DBS_TYPE</b>	Tipo.	Caracter

A posição do campo não leva em consideração os campos internos do Protheus (Recno e Deleted).

**Sintaxe:** DBFIELDINFO ( < nINFOTIPO > , < nCAMPO > )

**Parâmetros:**

<b>nINFOTIPO</b>	Tipo de informação a ser verificada (DBS_NAME, DBS_DEC, DBS_LEN e DBS_TYPE).
<b>nCAMPO</b>	Posição do campo a ser verificado.

**Retorno:**

<b>Indefinido</b>	Retorna NIL se não há tabela corrente ou a posição do campo especificado está inválida. Informação do campo Informação requisitada pelo usuário (pode ser de tipo numérico se for tamanho ou casas decimais, tipo caracter se for nome ou tipo).
-------------------	---

**Exemplo:**

```
USE Clientes NEW

DBFIELDINFO(DBS_NAME,1) // Retorno: Nome
DBFIELDINFO(DBS_TYPE,1) // Retorno: C
DBFIELDINFO(DBS_LEN,1) // Retorno: 10
DBFIELDINFO(DBS_DEC,1) // Retorno: 0
```

---

**DBFILTER()**

A função DBFILTER() é utilizada para verificar a expressão de filtro ativo na área de trabalho corrente.

**Sintaxe:** DBFILTER()

**Parâmetros:**

<b>Nenhum</b>	()
---------------	----

**Retorno:**

<b>Caracter</b>	Retorna a expressão do filtro ativo na área de trabalho atual. Caso não exista filtro ativo retorna "" (String vazia).
-----------------	--

**Exemplo:**

```
USE Cliente INDEX Ind1 NEW
SET FILTER TO Nome > "Jose"
DBFILTER() // retorna: Nome > "Jose"
SET FILTER TO Num < 1000
DBFILTER() // retorna: Num < 1000
```



## DBGOTO()

---

Move o cursor da área de trabalho ativa para o *record number* (recno) especificado, realizando um posicionamento direto, sem a necessidade uma busca (seek) prévio.

**Sintaxe:** DbGoto(nRecno)

### Parâmetros

<b>nRecno</b>	<i>Record number</i> do registro a ser posicionado.
---------------	---

### Exemplo:

```
DbSelectArea("SA1")
DbGoto(100) // Posiciona no registro 100

IF !EOF() // Se a área de trabalho não estiver em final de arquivo
    MsgInfo("Você está no cliente:" + A1_NOME)
ENDIF
```

## DBGOTOP()

---

Move o cursor da área de trabalho ativa para o primeiro registro lógico.

**Sintaxe:** DbGoTop()

### Parâmetros

<b>Nenhum</b>	()
---------------	----

### Exemplo:

```
nCount := 0 // Variável para verificar quantos registros há no
intervalo
DbSelectArea("SA1")
DbSetOrder(1) // A1_FILIAL + A1_COD + A1_LOJA
DbGoTop()

While !BOF() // Enquanto não for o início do arquivo
    nCount++ // Incrementa a variável de controle de registros no
intervalo
    DbSkip(-1)
End
```

```
MsgInfo("Existem :" + STRZERO(nCount,6) + " registros no intervalo").
```

```
// Retorno esperado :000001, pois o DbGoTop posiciona no primeiro  
registro.
```

## **DBGOBUTTON()**

---

Move o cursor da área de trabalho ativa para o último registro lógico.

### **Sintaxe: DbGoButton()**

#### **Parâmetros**

Nenhum	()
--------	----

#### **Exemplo:**

```
nCount := 0 // Variável para verificar quantos registros há no  
intervalo  
DbSelectArea("SA1")  
DbSetOrder(1) // A1_FILIAL + A1_COD + A1_LOJA  
DbGoButton()  
  
While !EOF() // Enquanto não for o início do arquivo  
    nCount++ // Incrementa a variável de controle de registros no  
    intervalo  
    DbSkip(1)  
End  
  
MsgInfo("Existem :" + STRZERO(nCount,6) + " registros no intervalo").  
  
// Retorno esperado :000001, pois o DbGoButton posiciona no último  
registro.
```

## **DBINFO()**

---

DBINFO() é utilizada para obter informações sobre a tabela corrente. O tipo de informação (primeiro argumento) é escolhido de acordo com as constantes abaixo:

**Tabela A : Constantes utilizadas na parametrização da função**

Constante	Descrição	Retorno
<b>DBI_GETRECSIZE</b>	Tamanho do registro em número de bytes similar a RECSIZE.	Numérico
<b>DBI_TABLEEXT</b>	Extensão do arquivo da tabela corrente.	Caracter
<b>DBI_FULLPATH</b>	Nome da tabela corrente com caminho completo.	Caracter
<b>DBI_BOF</b>	Se está posicionada no início da tabela similar a BOF	Lógico
<b>DBI_EOF</b>	Se está posicionada no final da tabela similar a EOF	Lógico
<b>DBI_FOUND</b>	Se a tabela está posicionada após uma pesquisa similar a FOUND	Lógico
<b>DBI_FCOUNT</b>	Número de campos na estrutura da tabela corrente similar a FCOUNT	Numérico
<b>DBI_ALIAS</b>	Nome do Alias da área de trabalho corrente similar a ALIAS	Caracter
<b>DBI_LASTUPDATE</b>	Data da última modificação similar a LUPDATE	Data

**Sintaxe:** DBINFO(<nINFOTIPO>)

**Parâmetros:**

<b>nINFOTIPO</b>	Tipo de informação a ser verificada.
------------------	--------------------------------------

**Retorno:**

<b>Indefinido</b>	Informação da Tabela Informação requisitada pelo usuário (o tipo depende da informação requisitada). Se não houver tabela corrente retorna NIL.
-------------------	---

**Exemplo:**

```
USE Clientes NEW
DBINFO(DBI_FULLPATH) // Retorno: C:\Teste\Clientes.dbf
DBINFO(DBI_FCOUNT) // Retorno: 12
DBGOTOP()
```

```

DBINFO(DBI_BOF) // Retorno: .F.
DBSKIP(-1)
DBINFO(DBI_BOF) // Retorno: .T.

```

## **DBNICKINDEXKEY()**

---

Função que retorna o “IndexKey”, ou seja, a expressão de índice da ordem especificada pelo NickName. Se não existe índice com o nickname, retorna uma *string* vazia.

**Sintaxe:** DBNICKINDEXKEY(<cNick>)

**Parâmetros:**

<b>cNick</b>	Indica o "NickName" da ordem de índice.
--------------	---

**Retorno:**

<b>Caracter</b>	Expressão do índice identificado pelo "NickName".
-----------------	---

## **DBORDERINFO()**

---

A função DBORDERINFO() é utilizada para obter informações sobre determinada ordem. A especificação da ordem pode ser realizada através de seu nome ou sua posição dentro da lista de ordens, mas se ela não for especificada serão obtidas informações da ordem corrente. O tipo de informação (primeiro argumento) é escolhido de acordo com as constantes abaixo:

**Tabela A : Constantes utilizadas na parametrização da função**

Constante	Descrição	Retorno
<b>DBOI_BAGNAME</b>	Nome do arquivo de índice ao qual a ordem pertence.	Caracter
<b>DBOI_FULLPATH</b>	do arquivo de índice (com seu diretório) ao qual a ordem pertence.	Caracter
<b>DBOI_ORDERCOUNT</b>	Número de ordens existentes no arquivo de índice especificado.	Caracter

**Sintaxe:** DBORDERINFO(<nINFOTIPO>)

**Parâmetros:**

<b>nINFOTIPO</b>	Nome do arquivo de índice.
------------------	----------------------------

**Retorno:**

<b>Caracter</b>	Retorna a informação da Ordem requisitada pelo usuário (pode ser de tipo numérico se for número de ordens no índice, tipo caracter se for nome do arquivo de índice). Caso não exista ordem corrente ou a posição da ordem especificada está inválida retorna NIL.
-----------------	--

**Exemplo:**

```
DBORDERINFO(DBOI_BAGNAME) // retorna: Ind  
DBORDERINFO(DBOI_FULLPATH) // retorna: C:\AP6\Teste\Ind.cdx
```

---

### **DBORDERNICKNAME()**

---

A função DBORDERNICKNAME() é utilizada para selecionar a ordem ativa através de seu apelido. Esta ordem é a responsável pela seqüência lógica dos registros da tabela corrente.

**Sintaxe: DBORDERNICKNAME(<cAPELIDO>)**

**Parâmetros:**

<b>cAPELIDO</b>	Nome do apelido da ordem a ser setada.
-----------------	--

**Retorno:**

<b>Lógico</b>	Retorna Falso se não conseguiu tornar a ordem ativa. Principais erros: Não existe tabela ativa ou não foi encontrada a ordem com o apelido. Retorna Verdadeiro se a ordem foi setada com sucesso.
---------------	--

**Exemplo:**

```
USE Cliente NEW  
SET INDEX TO Nome, Idade  
  
IF !DBORDERNICKNAME("IndNome")  
Messagebox("Registro não encontrado","Erro", 0)  
ENDIF
```

## **DBPACK()**

---

A função DBPACK() remove fisicamente todos os registros com marca de excluído da tabela.

**Sintaxe:** **\_\_DBPACK()**

**Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

**Retorno:**

<b>Nenhum</b>	( )
---------------	-----

**Exemplo:**

```
USE Clientes NEW
DBGOTO(100)
DBDELETE()
DBGOTO(105)
DBDELETE()
DBGOTO(110)
DBDELETE()

// Se a exclusão for confirmada:
__DBPACK()
```

## **DBRECALL()**

---

A função DBRECALL() é utilizada para retirar a marca de registro deletado do registro atual. Para ser executada o registro atual deve estar bloqueado ou a tabela deve estar aberta em modo exclusivo. Se o registro atual não estiver deletado, esta função não faz nada. Ela é o oposto da função DBDELETE() que marca o registro atual como deletado.

**Sintaxe:** **DBRECALL()**

**Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

**Retorno:**

<b>Nenhum</b>	( )
---------------	-----

### **Exemplo 01: Desfazendo a deleção do registro posicionado do alias corrente**

```
USE Cliente
DBGOTO(100)
DBDELETE()
DELETED() // Retorna: .T.
DBRECALL()
DELETED() // Retorna: .F.
```

### **Exemplo 02: Desfazendo as deleções do alias corrente**

```
USE Cliente
DBGOTOP()
WHILE !EOF()
    DBRECALL()
    DBSKIP()
ENDDO
```

### **DBRECORDINFO()**

A função DBRECORDINFO() é utilizada para obter informações sobre o registro especificado pelo segundo argumento (recno) da tabela corrente, se esta informação for omitida será verificado o registro corrente. O tipo de informação (primeiro argumento) é escolhido de acordo com as constantes abaixo:

**Tabela A : Constantes utilizadas na parametrização da função**

Constante	Descrição	Retorno
DBRI_DELETED	Estado de deletado similar a DELETED	Lógico
DBRI_RECSIZE	Tamanho do registro similar a RECSIZE	Numérico
DBRI_UPDATED	Verifica se o registro foi alterado e ainda não foi atualizado fisicamente similar a UPDATED	Lógico

**Sintaxe:** DBRECORDINFO (<nINFOTIPO> , [ nREGISTRO ] ) --> xINFO

**Parâmetros:**

nINFOTIPO	Tipo de informação a ser verificada.
nREGISTRO	Número do registro a ser verificado.

**Retorno:**

<b>Indefinido</b>	Não há tabela corrente ou registro inválido. Informação do Registro. Informação requisitada pelo usuário (o tipo depende da informação requisitada).
-------------------	--

**Exemplo:**

```
USE Clientes NEW
DBGOTO(100)
DBRECORDINFO(DBRI_DELETED) // Retorno: .F.
DBDELETE()
DBRECORDINFO(DBRI_DELETED) // Retorno: .F.
DBRECALL()
DBRECORDINFO(DBRI_RECSIZE) // Retorno: 230
NOME := "JOAO"
DBGOTO(200)
DBRECORDINFO(DBRI_UPDATED) // Retorno: .F.
DBRECORDINFO(DBRI_UPDATED,100) // Retorno: .T.
```

---

**DBREINDEX()**

A função DBREINDEX() reconstrói todos os índices da área de trabalho corrente e posiciona as tabelas no primeiro registro lógico.

**Sintaxe: DBREINDEX()**

**Parâmetros:**

<b>Nenhum</b>	()
---------------	----

**Retorno:**

<b>Nenhum</b>	()
---------------	----

**Exemplo:**

```
USE Clientes NEW
DBSETINDEX("IndNome")
DBREINDEX()
```

## **DBRLOCK()**

---

A função DBRLOCK() é utilizada quando se tem uma tabela aberta e compartilhada e se deseja bloquear um registro para que outros usuários não possam alterá-lo.

Se a tabela já está aberta em modo exclusivo, a função não altera seu estado.

O usuário pode escolher o registro a ser bloqueado através do parâmetro (recno), mas se este for omitido será bloqueado o registro corrente como na função RLOCK().

Esta função é o oposto à DBRUNLOCK, que libera registros bloqueados.

**Sintaxe: DBRLOCK([nREGISTRO])**

**Parâmetros:**

nREGISTRO	Número do registro a ser bloqueado.
-----------	-------------------------------------

**Retorno:**

Lógico	Retorna Falso se não conseguiu bloquear o registro. Principal motivo: o registro já foi bloqueado por outro usuário.  Retorna Verdadeiro se o registro foi bloqueado com sucesso.
--------	---

**Exemplo:**

```
DBUSEAREA( .T., "dbfcxdxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F.  
)  
DBGOTO(100)  
DBRLOCK() // Bloqueia o registro atual (100)  
DBRLOCK(110) // Bloqueia o registro de número 110
```

---

## **DBRLOCKLIST()**

---

A função DBRLOCKLIST() é utilizada para verificar quais registros estão locados na tabela corrente. Para tanto, é retornada uma tabela unidimensional com os números dos registros.

**Sintaxe: DBRLOCKLIST()**

**Parâmetros:**

Nenhum	()
--------	----

**Retorno:**

<b>Array</b>	Retorna NIL se não existe tabela corrente ou não existe nenhum registro locado. Retorna a lista com os recnos dos registros locados na tabela corrente.
--------------	---

**Exemplo:**

```
DBUSEAREA( .T., "dbfcxdads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F.  
)  
DBGOTOP()  
DBRLOCK() // Bloqueia o primeiro registro  
DBRLOCK(110) // Bloqueia o registro de número 110  
DBRLOCK(100) // Bloqueia o registro de número 100  
DBRLOCKLIST() // Retorna: {1,100,110}
```

---

### **DBRUNLOCK()**

A função DBRUNLOCK() é utilizada para liberar determinado registro bloqueado. O usuário pode escolher o registro a ser desbloqueado através do parâmetro (Recno), mas se este for omitido será desbloqueado o registro corrente como na função DBUNLOCK(). Esta função é o oposto à DBRLOCK, que bloqueia os registros.

**Sintaxe: DBRUNLOCK([nREGISTRO])**

**Parâmetros:**

<b>nREGISTRO</b>	Número do registro a ser desbloqueado.
------------------	--

**Retorno:**

<b>Nenhum</b>	()
---------------	----

**Exemplo:**

```
DBUSEAREA( .T., "dbfcxdads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F.  
)  
DBGOTO(100)  
DBRUNLOCK() //Desbloqueia o registro atual (100)  
DBRUNLOCK(110) // Desbloqueia o registro de número 110
```

## **DBSETDRIVER()**

---

A função DBSETDRIVER() pode ser utilizada apenas para verificar qual o RDD que está definido como padrão quando for omitido seu parâmetro. Ela também pode ser utilizada para especificar outro RDD como padrão, especificando-o através do parâmetro.

**Sintaxe:** **DBSETDRIVER([cNOVORDD])**

**Parâmetros:**

<b>cNOVORDD</b>	Novo nome do RDD a ser definido como padrão.
-----------------	--

**Retorno:**

<b>Caracter</b>	Nome do RDD padrão corrente.
-----------------	------------------------------

**Exemplo:**

```
DBSETDRIVER("CTREECDX") // Retorna: DBFCDX  
DBSETDRIVER() // Retorna: CTREECDX
```



*Importante*

Note que ao utilizar a função DBSETDRIVER para redefinir o driver corrente, o retorno da função não será o driver definido nos parâmetros, mas o driver que estava em uso antes da atualização.

---

## **DBSETINDEX()**

---

A função DBSETINDEX() é utilizada para acrescentar uma ou mais ordens de determinado índice na lista de ordens ativas da área de trabalho. Quando o arquivo de índice possui apenas uma ordem, a mesma é acrescentada à lista e torna-se ativa. Quando o índice possui mais de uma ordem, todas são acrescentadas à lista e a primeira torna-se ativa.



*Importante*

Para utilizar os arquivos de extensão padrão do RDD este dado deve ser especificado.

**Sintaxe:** **DBSETINDEX(<@cARQINDICE>)**

**Parâmetros:**

<b>cARQINDICE</b>	Nome do arquivo de índice, com ou sem diretório.
-------------------	--

**Retorno:**

Nenhum	()
--------	----

**Exemplo:**

USE Cliente NEW DBSETINDEX("Ind1") DBSETINDEX("\teste\Ind2.cdx")
--

**DBSETNICKNAME()**

---

A função DBSETNICKNAME() é utilizada para colocar um apelido em determinada ordem

especificada pelo primeiro parâmetro. Caso seja omitido o nome do apelido a ser dado, a função apenas verifica o apelido corrente.

**Sintaxe:** DBSETNICKNAME(<cINDICE>, [cAPELIDO])

**Parâmetros:**

cINDICE	Nome da ordem que deve receber o apelido.
---------	---

Caracter	Retorna "" (String vazia) se não conseguiu encontrar a ordem especificada, não conseguiu setar o apelido ou não havia apelido. Retorna o apelido corrente.
----------	--

**Exemplo:**

USE Cliente NEW DBSETNICKNAME("IndNome") // retorna: "" DBSETNICKNAME("IndNome","NOME") // retorna: "" DBSETNICKNAME("IndNome") // retorna: "NOME"
---

## **DBSELECTAREA()**

---

Define a área de trabalho especificada com sendo a área ativa. Todas as operações subsequentes que fizerem referência a uma área de trabalho a utilização, a menos que a área desejada seja informada explicitamente.

- Sintaxe:** DbSelectArea(**nArea** | **cArea**)

- Parâmetros**

<b>nArea</b>	Valor numérico que representa a área desejada, em função de todas as áreas já abertas pela aplicação, que pode ser utilizado ao invés do nome da área.
<b>cArea</b>	Nome de referência da área de trabalho a ser selecionada.

### **Exemplo 01: DbselectArea(nArea)**

```
nArea := Select("SA1") //          ↗ 10 (proposto)

DbSelectArea(nArea) // De acordo com o retorno do comando Select()

ALERT("Nome do cliente: "+A1_NOME) // Como o SA1 é o alias
selecionado, os comandos
    // a partir da seleção do alias compreendem que ele
    // está implícito na expressão, o que causa o mesmo
    // efeito de SA1->A1_NOME
```

### **Exemplo 02: DbselectArea(cArea)**

```
DbSelectArea("SA1") // Especificação direta do alias que deseja-se
selecionar

ALERT("Nome do cliente: "+A1_NOME) // Como o SA1 é o alias
selecionado, os comandos
    // a partir da seleção do alias compreendem que ele
    // está implícito na expressão, o que causa o mesmo
    // efeito de SA1->A1_NOME
```

## **DBSETORDER()**

---

Define qual índice será utilizado pela área de trabalho ativa, ou seja, pela área previamente selecionada através do comando DbSelectArea(). As ordens disponíveis no ambiente Protheus são aquelas definidas no SINDEX /SIX, ou as ordens disponibilizadas por meio de índices temporários.

**Sintaxe:** **DbSetOrder(nOrdem)**

**Parâmetros**

<b>nOrdem</b>	Número de referência da ordem que deseja ser definida como ordem ativa para a área de trabalho.
---------------	---

**Exemplo:**

```
DbSelectArea("SA1")
DbSetOrder(1) // De acordo com o arquivo SIX ->
A1_FILIAL+A1_COD+A1_LOJA
```

## **DBORDERNICKNAME()**

---

Define qual índice criado pelo usuário seja utilizado. O usuário pode incluir os seus próprios índices e no momento da inclusão deve criar o NICKNAME para o mesmo.

**Sintaxe:** **DbOrderNickName(NickName)**

**Parâmetros**

<b>NickName</b>	NickName atribuído ao índice criado pelo usuário.
-----------------	---

**Exemplo:**

```
DbSelectArea("SA1")
DbOrderNickName("Tipo") // De acordo com o arquivo SIX -> A1_FILIAL+A1_TIPO
NickName: Tipo
```

## DBSEEK() E MSSEEK()

**DbSeek():** Permite posicionar o cursor da área de trabalho ativo no registro com as informações especificadas na chave de busca, fornecendo um retorno lógico indicando se o posicionamento foi efetuado com sucesso, ou seja, se a informação especificada na chave de busca foi localizada na área de trabalho.

**Sintaxe:** DbSeek(**cChave, ISoftSeek, ILast**)

### Parâmetros

<b>cChave</b>	Dados do registro que deseja-se localizar, de acordo com a ordem de busca previamente especificada pelo comando DbSetOrder(), ou seja, de acordo com o índice ativo no momento para a área de trabalho.
<b>ISoftSeek</b>	Define se o cursor ficará posicionado no próximo registro válido, em relação a chave de busca especificada, ou em final de arquivo, caso não seja encontrada exatamente a informação da chave. Padrão <input checked="" type="checkbox"/> F.
<b>ILast</b>	Define se o cursor será posicionado no primeiro ou no último registro de um intervalo com as mesmas informações especificadas na chave. Padrão <input checked="" type="checkbox"/> F.

### Exemplo 01 – Busca exata

```
DbSelectArea("SA1")
DbSetOrder(1) // acordo com o arquivo SIX ->
A1_FILIAL+A1_COD+A1_LOJA

IF DbSeek("01" + "000001" + "02" ) // Filial: 01, Código: 000001,
Loja: 02

    MsgInfo("Cliente localizado", "Consulta por cliente")

Else
    MsgAlert("Cliente não encontrado", "Consulta por cliente")

Endif
```

### Exemplo 02 – Busca aproximada

```
DbSelectArea("SA1")
DbSetOrder(1) // acordo com o arquivo SIX ->
A1_FILIAL+A1_COD+A1_LOJA

DbSeek("01" + "000001" + "02", .T.) // Filial: 01, Código: 000001,
Loja: 02
```

```

// Exibe os dados do cliente localizado, o qual pode não ser o
especificado na chave:

MsgInfo("Dados do cliente localizado: "+CRLF +
        "Filial:" + A1_FILIAL + CRLF +
        "Código:" + A1_COD + CRLF +
        "Loja:" + A1_LOJA + CRLF +
        "Nome:" + A1_NOME + CRLF, "Consulta por cliente")

```

**MsSeek()**: Função desenvolvida pela área de Tecnologia da Microsiga, a qual possui as mesmas funcionalidades básicas da função DbSeek(), com a vantagem de não necessitar acessar novamente a base de dados para localizar uma informação já utilizada pela *thread* (conexão) ativa.

Desta forma, a *thread* mantém em memória os dados necessários para reposicionar os registros já localizados através do comando DbSeek (no caso o Recno()) de forma que a aplicação pode simplesmente efetuar o posicionamento sem executar novamente a busca.

A diferença entre o DbSeek() e o MsSeek() é notada em aplicações com grande volume de posicionamentos, como relatórios, que necessitam referenciar diversas vezes o mesmo registro durante uma execução.

## **DBSKIP()**

---

Move o cursor do registro posicionado para o próximo (ou anterior dependendo do parâmetro), em função da ordem ativa para a área de trabalho.

.     **Sintaxe: DbSkip(nRegistros)**

.     **Parâmetros**

<b>nRegistros</b>	Define em quantos registros o cursor será deslocado. Padrão 1
-------------------	---

### **Exemplo 01 – Avançando registros**

```

DbSelectArea("SA1")
DbSetOrder(2) // A1_FILIAL + A1_NOME
DbGotoP() // Posiciona o cursor no início da área de trabalho ativa

While !EOF() // Enquanto o cursor da área de trabalho ativa não
indicar fim de arquivo
    MsgInfo("Você está no cliente:" + A1_NOME)
    DbSkip()
End

```

## Exemplo 02 – Retrocedendo registros

```
DbSelectArea("SA1")
DbSetOrder(2) // A1_FILIAL + A1_NOME
DbGoBotton() // Posiciona o cursor no final da área de trabalho
ativa

While !BOF() // Enquanto o cursor da área de trabalho ativa não
indicar início de arquivo
    MsgInfo("Você está no cliente:" + A1_NOME)
    DbSkip(-1)
End
```

## DBSETFILTER()

Define um filtro para a área de trabalho ativa, o qual pode ser descrito na forma de um bloco de código ou através de uma expressão simples.

### Sintaxe: **DbSetFilter(bCondicao, cCondicao)**

#### Parâmetros

<b>bCondicao</b>	Bloco de expressa a condição de filtro em forma executável.
<b>cCondicao</b>	Expressão de filtro simples na forma de <i>string</i> .

## Exemplo 01 – Filtro com bloco de código

```
bCondicao := {|| A1_COD >= "000001" .AND. A1_COD <= "001000"}
DbSelectArea("SA1")
DbSetOrder(1)
DbSetFilter(bCondicao)
DbGoBotton()

While !EOF()
    MsgInfo("Você está no cliente:" + A1_COD)
    DbSkip()
End

// O último cliente visualizado deve ter o código menor do que
"001000".
```

## Exemplo 02 – Filtro com expressão simples

```
cCondicao := "A1_COD >= '000001' .AND. A1_COD <= '001000'"  
DbSelectArea("SA1")  
DbSetOrder(1)  
DbSetFilter(,cCondicao)  
DbGoBotton()  
  
While !EOF()  
    MsgInfo("Você está no cliente:" + A1_COD)  
    DbSkip()  
End  
  
// O último cliente visualizado deve ter o código menor do que  
"001000".
```

## DBSTRUCT()

Retorna um *array* contendo a estrutura da área de trabalho (alias) ativo. A estrutura será um *array* bidimensional conforme abaixo:

ID*	Nome campo	Tipo campo	Tamanho	Decimais
-----	------------	------------	---------	----------

\*Índice do *array*

**Sintaxe: DbStruct()**

**Parâmetros**

Nenhum	()
--------	----

**Exemplo:**

```
cCampos := ""  
DbSelectArea("SA1")  
aStructSA1 := DbStruct()  
  
FOR nX := 1 to Len(aStructSA1)  
  
    cCampos += aStructSA1[nX][1] + "/"  
  
NEXT nX  
  
ALERT(cCampos)
```

## **DBUNLOCK()**

---

A função DBUNCLOK() retira os bloqueios dos registros e do arquivo da tabela corrente.

**Sintaxe: DBUNLOCK()**

**Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

**Retorno:**

<b>Nenhum</b>	( )
---------------	-----

## **DBUNLOCKALL()**

---

A função DBUNLOCKALL() Retira os bloqueios de todos os registros e dos arquivos de todas as tabelas abertas. Esta função é utilizada para liberar todos os registros bloqueados e é equivalente a executar DBUNLOCK para todas as tabelas da área de trabalho.

**Sintaxe: DBUNLOCKALL()**

**Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

**Retorno:**

<b>Nenhum</b>	( )
---------------	-----

## **DBUSEAREA()**

---

Define um arquivo de base de dados como uma área de trabalho disponível na aplicação.

**Sintaxe: DbUseArea(lNovo, cDriver, cArquivo, cAlias, lComparilhado,; lSoLeitura)**

### Parâmetros

<b>INovo</b>	Parâmetro opcional que permite que se caso o cAlias especificado já esteja em uso, ele seja fechado antes da abertura do arquivo da base de dados.
<b>cDriver</b>	Driver que permita a aplicação manipular o arquivo de base de dados especificado. A aplicação ERP possui a variável __LOCALDRIVER definida a partir das configurações do .ini do server da aplicação. Algumas chaves válidas: "DBFCDX", "CTREECDX", "DBFCDXAX", "TOPCONN".
<b>cArquivo</b>	Nome do arquivo de base de dados que será aberto com o alias especificado.
<b>cAlias</b>	Alias para referência do arquivos de base de dados pela aplicação.
<b>IComparilhado</b>	Se o arquivo poderá ser utilizado por outras conexões.
<b>ISoLeitura</b>	Se o arquivo poderá ser alterado pela conexão ativa.

### Exemplo:

```
DbUserArea(.T., "DBFCDX", "\SA1010.DBF", "SA1DBF", .T., .F.)  
DbSelectArea("SA1DBF")  
MsgInfo("A tabela SA1010.DBF possui:" + STRZERO(RecCount(),6) + "  
registros.")  
DbCloseArea()
```

### **DELETED()**

A função DELETED() Verifica se o registro está com marca de excluído. Quando o registro é excluído, permanece fisicamente na tabela, mas fica marcado como excluído. Esta função verifica este estado. Se nenhuma área está selecionada, retorna .F.. Quando é executada a função DBPACK() todos os registros marcados como deletados são apagados fisicamente. A função DBRECALL() retira todas as marcas.

#### Sintaxe: DELETED()

#### Parâmetros:

<b>Nenhum</b>	()
---------------	----

#### Retorno:

<b>Nenhum</b>	()
---------------	----

**Exemplo:**

```
USE "\DADOSADV\AA1990.DBF" SHARED NEW  
DBGOTO(100)  
IF DELETED()  
Messagebox("O registro atual foi deletado","Erro", 0)  
ENDIF
```

**FCOUNT()**

A função FCOUNT() avalia a quantidade de campos existentes na estrutura do arquivo ativo como área de trabalho.

**Sintaxe: FCOUNT()**

**Parâmetros:**

Nenhum	()
--------	----

**Retorno:**

Numérico	Quantidade de campos existentes na estrutura da área de trabalho ativa.
----------	---

**Exemplo:**

```
DbSelectArea("SA1")  
nFields := FCOUNT()  
  
IF nFields > 0  
    MSGINFO("A estrutura da tabela contém  
    :+CvalToChar(nFields)+"campos.")  
ENDIF
```

## **FOUND()**

---

A função FOUND() recupera o resultado de sucesso referente a última operação de busca efetuada pelo processamento corrente.

**Sintaxe: FOUND()**

**Parâmetros:**

<b>Nenhum</b>	<b>()</b>
---------------	-----------

**Retorno:**

<b>Lógico</b>	Indica se a última operação de busca realizada pelo processamento corrente obteve sucesso (.T.) ou não (.F.).
---------------	---

**Exemplo:**

```
Pergunte(cPerg,T.)
DbSelectArea("SA1")
DbSetOrder(1)
DbSeek(xFilial("SA1")+MVPAR01)

IF Found()
    MSGINFO("Cliente encontrado")
ELSE
    MSGALERT("Dados não encontrados")
ENDIF
```

---

## **INDEXKEY()**

A função INDEXKEY() determina a expressão da chave de um índice especificado na área de trabalho corrente, e o retorna na forma de uma cadeia de caracteres, sendo normalmente utilizada na área de trabalho correntemente selecionada.

**Sintaxe: INDEXKEY()**

**Parâmetros:**

<b>nOrdem</b>	Ordem do índice na lista de índices abertos pelo comando USE...INDEX ou SET INDEX TO na área de trabalho corrente. O valor <i>default</i> zero especifica o índice corrente, independentemente de sua posição real na lista.
---------------	--

**Retorno:**

<b>Caracter</b>	Expressão da chave do índice especificado na forma de uma cadeia de caracteres. Caso não haja um índice correspondente, INDEXKEY() retorna uma cadeia de caracteres vazia ("").
-----------------	---

**Exemplo:**

```
cExpressao := SA1->(IndexKey())
```

---

## **INDEXORD()**

---

A função INDEXORD() verifica a posição do índice corrente na lista de índices do respectivo alias.

**Sintaxe: INDEXORD()**

**Parâmetros:**

<b>Nenhum</b>	()
---------------	----

---

**Retorno:**

<b>Numérico</b>	Posição do índice corrente na lista de índices da tabela. Retorna 0 se não existe índice aberto na tabela corrente.
-----------------	---

**Exemplo:**

```
USE Cliente NEW
SET INDEX TO Nome, End, Cep
nOrd:=INDEXORD() // Return: 1 - é o primeiro índice da lista
```

## **LUPDATE()**

---

A função LUPDATE() verifica qual a data da última modificação e fechamento da tabela corrente, sendo que caso não exista tabela corrente é retornada uma data em branco.

### **Sintaxe: LUPDATE()**

#### **Parâmetros:**

<b>Nenhum</b>	<b>()</b>
---------------	-----------

#### **Retorno:**

<b>Data</b>	Retorna um valor do tipo Data , indicando a data da ultima modificação e fechamento da Tabela. Caso não haja tabela selecionada na área de trabalho atual , a função retornará uma data vazia (ctod("")) .
-------------	--

#### **Exemplo:**

```
// Mostra a data da última modificação da tabela corrente,  
dModificacao := LUpdate()  
IF (EMPTY(dModificacao))  
    CONOUT("Não há tabela corrente")  
ELSE  
    CONOUT(("Data da ultima modificacao : " + DTOS(dModificacao)))  
ENDIF
```

## **MSAPPEND()**

---

A função MsAppend() adiciona registros de um arquivo para outro, respeitando a estrutura das tabelas.

### **Sintaxe: MSAPPEND( [cArqDest], cArqOrig )**

#### **Parâmetros:**

<b>cArqDest</b>	Se o RDD corrente for DBFCDX os registros serão adicionados na área selecionada, caso contrário o arquivo destino terá que ser informado.
<b>cArqOrig</b>	Nome do arquivo origem contendo os registros a serem adicionados.

**Retorno:**

<b>Lógico</b>	Se a operação for realizada com sucesso o função retornará verdadeiro (.T.).
---------------	--

**Exemplo:**

```
dbSelectArea('XXX')
MsAppend(,'ARQ00001')
```

**MSUNLOCK()**

Libera o travamento (lock) do registro posicionado confirmando as atualizações efetuadas neste registro.

**Sintaxe: MsUnLock()**

**Parâmetros**

<b>Nenhum</b>	()
---------------	----

**Exemplo:**

```
DbSelectArea("SA1")
DbSetOrder(1) // A1_FILIAL + A1_COD + A1_LOJA
DbSeek("01" + "900001" + "01") // Busca exata

IF Found() // Avalia o retorno do último DbSeek realizado
  RecLock("SA1",.F.)
  SA1->A1_NOME := "CLIENTE CURSO ADVPL BÁSICO"
  SA1->A1_NREDUZ := "ADVPL BÁSICO"
  MsUnLock() // Confirma e finaliza a operação
ENDIF
```

## **ORDBAGEXT()**

---

A função ORDBAGEXT é utilizada no gerenciamento de indices para os arquivos de dados do sistema, permitindo avaliar qual a extensão deste índices atualmente em uso, de acordo com a RDD ativa.

**Sintaxe: ORDBAGEXT()**

**Parâmetros:**

<b>Nenhum</b>	()
---------------	----

**Retorno:**

<b>cBagExt</b>	Extensão do arquivo dos arquivos de índices em uso pelo sistema, determinado pela RDD ativa.
----------------	--

**Exemplo:**

```
cArqTRB := CriaTrab(aStruTRB,.T.)
// RDD UTILIZADA: "DBFCDXADS"
DbUseArea(.T., "DBFCDXADS", cArqTRB, "TRBSA1", .F., .F.)

DbSelectArea("TRBSA1")
cArqInd := CriaTrab(Nil,.F.)
IndRegua("TRBSA1",cArqInd,cChaveInd,"","Selecionando registros
...")
#IFNDEF TOP
    DbSetIndex(cArqInd+OrdBagExt())
// RETORNO: ".CDX"
#endif
DbSetOrder(1)
```

---

## **ORDKEY()**

A função ORDKEY() verifica qual é a expressão de chave de determinada ordem. Caso não sejam especificados os parâmetros de identificação da ordem, é verificada a ordem corrente. Para evitar conflito, no caso de haver mais de uma ordem com o mesmo nome, pode-se passar o parâmetro com o nome do índice ao qual a ordem pertence.

A ordem passada no primeiro parâmetro pode ser especificada através da sua posição na lista de ordens ativas (através do ORDLISTADD) ou através do nome dado à ordem. A função verifica automaticamente se o parâmetro é numérico ou caracter.

**Sintaxe: ORDKEY([cOrdem | nPosicao] , [cArqIndice])**

**Parâmetros:**

<b>cOrdem</b>	Há duas opções para o primeiro parâmetro: cNome: tipo caracter, contém nome do índice.
<b>nPosicao</b>	nPosicao: tipo numérico, indica ordem do índice.
<b>cArqIndice</b>	Nome do arquivo de índice.

**Retorno:**

<b>Caracter</b>	Expressão de chave da ordem ativa ou especificada pelos parâmetros. Cadeia vazia indica que não existe ordem corrente.
-----------------	---

**Exemplo:**

```
USE Cliente NEW
INDEX ON Nome+Cod TO Ind1 FOR Nome+Cod > 'AZZZZZZZ'
ORDKEY('Ind1')
// Retorna: Nome+Cod
```

---

**RECLOCK()**

Efetua o travamento do registro posicionado na área de trabalho ativa, permitindo a inclusão ou alteração das informações do mesmo.

**Sintaxe: RecLock(cAlias,lInclui)**

**Parâmetros**

<b>cAlias</b>	Alias que identifica a área de trabalho que será manipulada.
<b>lInclui</b>	Define se a operação será uma inclusão (.T.) ou uma alteração (.F.)

**Exemplo 01 - Inclusão**

```
DbSelectArea("SA1")
RecLock("SA1",.T.)
SA1->A1_FILIAL := xFilial("SA1") // Retorna a filial de acordo com
as configurações do ERP
SA1->A1_COD := "900001"
SA1->A1_LOJA := "01"
MsUnLock() // Confirma e finaliza a operação
```

## Exemplo 02 - Alteração

```
DbSelectArea("SA1")
DbSetOrder(1) // A1_FILIAL + A1_COD + A1_LOJA
DbSeek("01" + "900001" + "01") // Busca exata

IF Found() // Avalia o retorno do último DbSeek realizado
    RecLock("SA1",.F.)
    SA1->A1_NOME := "CLIENTE CURSO ADVPL BÁSICO"
    SA1->A1_NREDUZ := "ADVPL BÁSICO"
    MsUnLock() // Confirma e finaliza a operação
ENDIF
```

A linguagem ADVPL possui variações da função RecLock(), as quais são:



*Dica*

- . RLOCK()
- . DBRLOCK()

A sintaxe e a descrição destas funções estão disponíveis no Guia de Referência Rápido ao final deste material.

A linguagem ADVPL possui variações da função MsUnlock(), as quais são:



*Dica*

- . UNLOCK()
- . DBUNLOCK()
- . DBUNLOCKALL()

A sintaxe e a descrição destas funções estão disponíveis no Guia de Referência Rápido ao final deste material.

## **RECNO()**

---

A função RECNO() retorna o número do registro atualmente posicionado na área de trabalho ativa.

**Sintaxe:** RECNO()

**Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

**Retorno:**

<b>nRecno</b>	Identificador numérico do registro atualmente posicionando na área de trabalho ativa.
---------------	---

**Exemplo:**

```
DbSelectArea("SA1")
DbGoto(100) // Posiciona no registro de recno 100.
MSGINFO("Registro posicionado:" + cValToChar(RECNO()))
```

## **SELECT()**

---

A função SELECT() determina o número da área de trabalho de um alias. O número retornado pode variar de zero a 250. Se <cAlias> não for especificado, é retornado o número da área de trabalho corrente. Caso <cAlias> seja especificado e o alias não existir, SELECT() retorna zero.

**Sintaxe:** SELECT([cAlias])

**Parâmetros:**

<b>cAlias</b>	Nome da área de trabalho a ser verificada.
---------------	--

**Retorno:**

<b>Numérico</b>	Área de trabalho do alias especificado na forma de um valor numérico inteiro.
-----------------	---

**Exemplo:**

```
nArea := Select("SA1")
ALERT("Referência do alias SA1: " + STRZERO(nArea,3)) //  
(proposto)
```

## SET FILTER TO

---

O comando SET FILTER TO define uma condição de filtro que será aplicada a área de trabalho ativa.

---



*Importante*

Recomenda-se o uso da função DbSetFilter() em substituição ao comando SET FILTER TO

---

### Sintaxe: SET FILTER TO cCondicao

#### Parâmetros:

<b>cCondicao</b>	Expressão que será avaliada pela SET FILTER, definindo os registros que ficarão disponíveis na área de trabalho ativa. Esta expressão obrigatoriamente deve ter um retorno lógico.
------------------	---

#### Retorno:

<b>Nenhum</b>	()
---------------	----



*Dica*

O uso da sintaxe SET FILTER TO desativa o filtro na área de trabalho corrente.

---

#### Exemplo:

```
USE Employee INDEX Name NEW
```

```
SET FILTER TO Age > 50
```

```
LIST LastName, FirstName, Age, Phone
```

```
SET FILTER TO
```

## **SOFTLOCK()**

---

Permite a reserva do registro posicionado na área de trabalho ativa de forma que outras operações, com exceção da atual, não possam atualizar este registro. Difere da função RecLock() pois não gera uma obrigação de atualização, e pode ser sucedido por ele.

Na aplicação ERP Protheus, o SoftLock() é utilizado nos browses, antes da confirmação da operação de alteração e exclusão, pois neste momento a mesma ainda não foi efetivada, mas outras conexões não podem acessar aquele registro pois o mesmo está em manutenção, o que implementa da integridade da informação.

### **Sintaxe: SoftLock(cAlias)**

#### **Parâmetros**

<b>cAlias</b>	Alias de referência da área de trabalho ativa, para o qual o registro posicionado será travado.
---------------	---

#### **Exemplo:**

```
cChave := GetCliente() // Função ilustrativa que retorna os dados de
busca de um cliente

DbSelectArea("SA1")
DbSetOrder(1)
DbSeek(cChave)

IF Found()
    SoftLock() // Reserva o registro localizado
    IConfirma := AlteraSA1() // Função ilustrativa que exibe os dados do
    registro
                                // posicionado e
    permite a alteração dos mesmos.

    IF IConfirma
        RecLock("SA1",.F.)
        GravaSA1() // Função ilustrativa que altera os dados conforme a
        AlertaSA1()
        MsUnLock() // Liberado o RecLock() e o SoftLock() do registro.
    Endif
Endif
```

## **USED()**

---

A função USED() é utilizada para determinar se há um arquivo de banco de dados em uso em uma área de trabalho específica. O padrão é que USED() opere na área de trabalho correntemente selecionada.

### **Sintaxe: USED()**

#### **Parâmetros:**

<b>Nenhum</b>	()
---------------	----

#### **Retorno:**

<b>Lógico</b>	Verdadeiro (.T.) caso haja um arquivo de banco de dados em uso; caso contrário, retorna falso (.F.).
---------------	--

#### **Exemplo:**

```
USE Customer NEW
CONOUT(USED()) // Resulta: .T.
CLOSE
CONOUT (USED()) // Resulta: .F.
```

## **ZAP**

---

O comando ZAP remove fisicamente todos os registro da tabela corrente. É necessário que o alias em questão seja aberto em modo exclusivo para esta operação ser realizada.

### **Sintaxe: ZAP**

#### **Parâmetros:**

<b>Nenhum</b>	()
---------------	----

#### **Retorno:**

<b>Nenhum</b>	()
---------------	----

**Exemplo:**

```
cTabela := RetSqlName("SA4")
cAlias := "TMP"
USE (cTabela) ALIAS (cAlias) EXCLUSIVE NEW VIA "TOPCONN"
IfNetErr()
    MsgStop("Nao foi possivel abrir "+cTabela+" em modo
EXCLUSIVO.")
Else
    ZAP
    USE
    MsgStop("Registros da tabela "+cTabela+" eliminados com sucesso.")
Endif
```

## Controle de numeração seqüencial

**GETSXENUM()**

---

Obtém o número seqüência do alias especificado no parâmetro, através da referência aos arquivos de sistema SXE/SXF ou ao servidor de numeração, quando esta configuração está habilitada no ambiente Protheus.

**Sintaxe: GETSXENUM(cAlias, cCampo, cAliasSXE, nOrdem)**

**Parâmetros**

<b>cAlias</b>	Alias de referência da tabela para a qual será efetuado o controle da numeração seqüencial.
<b>cCampo</b>	Nome do campo no qual está implementado o controle da numeração.
<b>cAliasSXE</b>	Parâmetro opcional, quando o nome do alias nos arquivos de controle de numeração não é o nome convencional do alias para o sistema ERP.
<b>nOrdem</b>	Número do índice para verificar qual a próxima ocorrência do número.

**CONFIRMSXE()**

---

Confirma o número alocado através do último comando GETSXENUM().

**Sintaxe: CONFIRMSXE(lVerifica)**

**Parâmetros**

<b>lVerifica</b>	Verifica se o número confirmado não foi alterado, e por consequência já existe na base de dados.
------------------	--

## **ROLLBACKSXE()**

---

Descarta o número fornecido pelo último comando GETSXENUM(), retornando a numeração disponível para outras conexões.

### **Sintaxe: ROLLBACKSXE()**

#### **Parâmetros**

<b>Nenhum</b>	( )
---------------	-----

#### **Validação**

---

## **ALLWAYSFALSE()**

---

A função AllwaysFalse() foi criada com o objetivo de compatibilidade, sendo que sempre irá retornar um valor lógico falso, facilitando a especificação desta situação nas parametrizações de validações de modelos de interface pré-definidos no sistema.

### **Sintaxe: ALLWAYSFALSE()**

#### **Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

#### **Retorno:**

<b>Lógico</b>	Retorna um valor lógico sempre falso.
---------------	---------------------------------------

## **ALLWAYSTRUE()**

---

A função AllwaysTrue() foi criada com o objetivo de compatibilidade, sendo que sempre irá retornar um valor lógico verdadeiro, facilitando a especificação desta situação nas parametrizações de validações de modelos de interface pré-definidos no sistema.

### **Sintaxe: ALLWAYSTRUE()**

#### **Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

#### **Retorno:**

<b>Lógico</b>	Retorna um valor lógico sempre verdadeiro.
---------------	--

## **EXISTCHAV()**

---

Retorna .T. ou .F. se o conteúdo especificado existe no alias especificado. Caso exista será exibido um help de sistema com um aviso informando da ocorrência.

Função utilizada normalmente para verificar se um determinado código de cadastro já existe na tabela na qual a informação será inserida, como por exemplo o CNPJ no cadastro de clientes ou fornecedores.

**Sintaxe:** ExistChav(**cAlias**, **cConteudo**, **nÍndice**)

### **Parâmetros**

<b>cAlias</b>	Alias de referência para a validação da informação.
<b>cConteudo</b>	Chave a ser pesquisada, sem a filial.
<b>nÍndice</b>	Índice de busca para consulta da chave.

## **EXISTCPO()**

---

Retorna .T. ou .F. se o conteúdo especificado não existe no alias especificado. Caso não exista será exibido um help de sistema com um aviso informando da ocorrência.

Função utilizada normalmente para verificar se a informação digitada em um campo, a qual depende de outra tabela, realmente existe nesta outra tabela. Como, por exemplo, o código de um cliente em um pedido de venda.

**Sintaxe:** ExistCpo(**cAlias**, **cConteudo**, **nÍndice**)

### **Parâmetros**

<b>cAlias</b>	Alias de referência para a validação da informação.
<b>cConteudo</b>	Chave a ser pesquisada, sem a filial.
<b>nÍndice</b>	Índice de busca para consulta da chave.

## **LETTERORNUM()**

---

A função LETTERORNUM() avalia se um determinado conteúdo é composto apenas de letras e números (alfanumérico).

**Sintaxe:** LETTERORNUM(**cString**)

### **Parâmetros:**

<b>cString</b>	<i>String</i> que terá seu conteúdo avaliado.
----------------	---

**Retorno:**

Lógico	Indica que se a <i>string</i> avaliada contém apenas letras e número, ou seja, alfanumérico.
--------	--

## **NAOVAZIO()**

---

Retorna .T. ou .F. se o conteúdo do campo posicionado no momento não está vazio.

**Sintaxe: NaoVazio()**

**Parâmetros**

Nenhum	()
--------	----

## **NEGATIVO()**

---

Retorna .T. ou .F. se o conteúdo digitado para o campo é negativo.

**Sintaxe: Negativo()**

**Parâmetros**

Nenhum	()
--------	----

## **PERTENCE()**

---

Retorna .T. ou .F. se o conteúdo digitado para o campo está contido na *string* definida como parâmetro da função. Normalmente utilizada em campos com a opção de combo. Caso contrário, seria utilizada a função ExistCpo().

**Sintaxe: Pertence(cString)**

**Parâmetros**

cString	<i>String</i> contendo as informações válidas que podem ser digitadas para um campo.
---------	--

## **POSITIVO()**

---

Retorna .T. ou .F. se o conteúdo digitado para o campo é positivo.

**Sintaxe: Positivo()**

**Parâmetros**

<b>Nenhum</b>	( )
---------------	-----

## **TEXTO()**

---

Retorna .T. ou .F. se o conteúdo digitado para o campo contém apenas números ou alfanuméricos.

**Sintaxe: Texto()**

**Parâmetros**

<b>Nenhum</b>	( )
---------------	-----

## **VAZIO()**

---

Retorna .T. ou .F. se o conteúdo do campo posicionado no momento está vazio.

**Sintaxe: Vazio()**

**Parâmetros**

<b>Nenhum</b>	( )
---------------	-----

## **Manipulação de parâmetros do sistema**

## **GETMV()**

---

Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Caso o parâmetro não exista, será exibido um help do sistema informando a ocorrência.

**Sintaxe: GETMV(cParametro)**

**Parâmetros**

<b>cParametro</b>	Nome do parâmetro do sistema no SX6, sem a especificação da filial de sistema.
-------------------	--

## **GETNEWPAR()**

---

Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Caso o parâmetro não exista será exibido um help do sistema informando a ocorrência.

Difere do SuperGetMV() pois considera que o parâmetro pode não existir na versão atual do sistema, e por consequência não será exibida a mensagem de help.

**Sintaxe: GETNEWPAR(cParametro, cPadrao, cFilial)**

**Parâmetros**

<b>cParametro</b>	Nome do parâmetro do sistema no SX6, sem a especificação da filial de sistema.
<b>cPadrao</b>	Conteúdo padrão que será utilizado caso o parâmetro não exista no SX6.
<b>cFilial</b>	Define para qual filial será efetuada a consulta do parâmetro. Padrão é filial corrente da conexão.

## **PUTMV()**

---

Atualiza o conteúdo do parâmetro especificado no arquivo SX6, de acordo com as parametrizações informadas.

**Sintaxe:** **PUTMV(cParametro, cConteudo)**

### **Parâmetros**

<b>cParametro</b>	Nome do parâmetro do sistema no SX6, sem a especificação da filial de sistema.
<b>cConteudo</b>	Conteúdo que será atribuído ao parâmetro no SX6.

## **SUPERGETMV()**

---

Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Caso o parâmetro não exista, será exibido um help do sistema informando a ocorrência.

Difere do GetMv() pois os parâmetros consultados são adicionados em uma área de memória, para que em uma nova consulta não seja necessário acessar e pesquisar o parâmetro na base de dados.

**Sintaxe:** **SUPERGETMV(cParametro , lHelp , cPadrao , cFilial)**

### **Parâmetros**

<b>cParametro</b>	Nome do parâmetro do sistema no SX6, sem a especificação da filial de sistema.
<b>lHelp</b>	Se será exibida a mensagem de Help caso o parâmetro não seja encontrado no SX6.
<b>cPadrao</b>	Conteúdo padrão que será utilizado caso o parâmetro não exista no SX6.
<b>cFilial</b>	Define para qual filial será efetuada a consulta do parâmetro. Padrão é filial corrente da conexão.

## Controle de impressão

### AVALIMP()

A função AVALIMP() é utilizada em relatórios específicos em substituição da função CABEC(), configurando a impressora de acordo com o driver escolhido e os parâmetros de impressão disponíveis no array aReturn, respeitando o formato utilizado pela função SETPRINT().

#### Sintaxe: AVALIMP(**nLimite**)

##### Parâmetros:

<b>nLimite</b>	Tamanho do relatório em colunas, podendo assumir os valores 80,132 ou 220 colunas, respectivamente para os formatos “P”, “M” ou “G” de impressão.
----------------	---

##### Retorno:

<b>Caracter</b>	<i>String</i> com caracteres de controle, dependente das configurações escolhidas pelo usuário e do arquivo de driver especificado.
-----------------	---

#### Exemplo:

```
/*
+-----
| Função      | XAVALIMP     | Autor | ARNALDO RAYMUNDO JR. | Data |
01.01.2007 |
+-----
| Descrição    | Exemplo de utilização da função AXCADASTRO()   |
|+
| Uso         | Curso ADVPL           |
|+
*/
USER FUNCTION XAVALIMP()

LOCAL cTitulo      := PADC("AVALIMP",74)
LOCAL cDesc1       := PADC("Demonstração do uso da função
AVALIMP()",74)
LOCAL cDesc2       := ""
LOCAL cDesc3       := PADC("CURSO DE ADVPL",74)
LOCAL cTamanho     := "G"
LOCAL cLimite      := 220
LOCAL cNatureza   := ""
LOCAL aReturn      := {"Especial", 1,"Administração", 1, 2, 2,"",1}
LOCAL cNomeProg   := "RAVALIMP"
LOCAL cPerg        := PADR("RAVALIMP",10) // Compatibilização
com MP10
```

```

LOCAL nLastKey      := 0
LOCAL cString       := "SF2"

Pergunte(cPerg,.F.) // Pergunta no SX1

wnrel:=
SetPrint(cString,wnrel,cPerg,cTitulo,cDesc1,cDesc2,cDesc3,.T.)

SetDefault(aReturn,cString)

If nLastKey == 27
    Return
Endif

RptStatus({| | RunReport(cString)},cTitulo)
Return

/*
+-----
| Função      | RUNREPORT   | Autor | ----- | Data |
01.01.2007 |
+-----
| Descrição          | Função interna de processamento utilizada pela
XAVALIMP()           |
|+-----
| Uso            | Curso ADVPL           |
|+-----
*/
Static Function RunReport(cString)
SetPrc(0,0)

//+-----+
//| Chamada da função AVALIMP()           |
//+-----+
@ 00,00 PSAY AvalImp(220)
dbSelectArea(cString)
dbSeek(xFilial()+mv_par01+mv_par03,.T.)
...
Return

```

## **CABEC()**

---

A função CABEC() determina as configurações de impressão do relatório e imprime o cabeçalho do mesmo.

**Sintaxe:** Cabec(**cTitulo**, **cCabec1**, **cCabec2**, **cNomeProg**, **nTamanho**,  
**nCompress**, **aCustomText**, **IPerg**, **cLogo**)

**Parâmetros:**

<b>cTitulo</b>	Título do relatório.
<b>cCabec1</b>	<i>String</i> contendo as informações da primeira linha do cabeçalho.
<b>cCabec2</b>	<i>String</i> contendo as informações da segunda linha do cabeçalho.
<b>cNomeProg</b>	Nome do programa de impressão do relatório.
<b>nTamanho</b>	Tamanho do relatório em colunas (80, 132 ou 220).
<b>nCompress</b>	Indica se impressão será comprimida (15) ou normal (18).
<b>aCustomText</b>	Texto específico para o cabeçalho, substituindo a estrutura padrão do sistema.
<b>IPerg</b>	Permite a supressão da impressão das perguntas do relatório, mesmo que o parâmetro MV_IMPSX1 esteja definido como "S".

**Parâmetros (continuação):**

<b>cLogo</b>	Redefine o bitmap que será impresso no relatório, não necessitando que ele esteja no formato padrão da Microsiga: "LGRL"+SM0->M0_CODIGO+SM0->M0_CODFIL+.BMP"
--------------	---

**Retorno:**

<b>Nenhum</b>	()
---------------	----

**Exemplo:**

```
#INCLUDE "protheus.ch"

/*
+-----
| Função      | MPTR001      | Autor | ARNALDO RAYMUNDO JR. | Data |
01.01.2007 |
+-----
| Descrição    | Exemplo de utilização das funções de impressão
CABEC()       |
|+
| Uso          | Curso ADVPL
|+
*/
/*
```





```

cCabec2 := "ESTADO"+Space(2)+"MUNICIPIO"+Space(8)+"ENDERECO"

dbSelectArea("TRBSA1")
dbGoTop()

SetRegua(RecCount())

While !EOF()

    If lAbortPrint .OR. nLastKey == 27
        @nLin,00 PSAY "*** CANCELADO PELO OPERADOR ***"
        Exit
    Endif

    If nLin > 55 // Salto de Página. Neste caso o formulario tem 55
    linhas...
        Cabec(cTitulo,cCabec1,cCabec2,cNomeProg,cTamanho,nTipo)
        nLin := 9
    Endif
    ...

```

## **IMPCADAST()**

---

A função IMPCADAST() cria uma interface simples que permite a impressão dos cadastros do sistema com parametrização DE/ATE.

**Sintaxe: IMPCADAST(cCab1, cCab2, cCab3, cNomeProg, cTam, nLimite, cAlias)**

**Parâmetros:**

<b>cCab1</b>	Primeira linha do cabeçalho.
<b>cCab2</b>	Segunda linha do cabeçalho.
<b>cCab3</b>	Terceira linha do cabeçalho.
<b>cNomeProg</b>	Nome do programa.
<b>cTam</b>	Tamanho do relatório nos formatos "P", "M" e "G".
<b>nLimite</b>	Número de colunas do relatório, seguindo o formato especificado no tamanho, aonde: "P"- 80 colunas "M"- 132 colunas "G"- 220 colunas
<b>cAlias</b>	Alias do arquivo de cadastro que será impresso.

**Retorno:**

Nenhum	()
--------	----

**MS\_FLUSH()**

---

A função MS\_FLUSH() envia o spool de impressão para o dispositivo previamente especificado com a utilização das funções AVALIMP() ou SETPRINT().

**Sintaxe: MS\_FLUSH()**

**Parâmetros:**

Nenhum	()
--------	----

**Retorno:**

Nenhum	()
--------	----

**Exemplo:**

```
/*
-----+
| Função      | RUNREPORT   | Autor | ----- | Data |
01.01.2007 |
+-----+
| Descrição          | Função interna de processamento utilizada pela
MPTR001()           |
|+-----+
| Uso      | Curso ADVPL |
|+-----+
| Observação| Continuação do exemplo da função CABEC()
|+-----+
*/
Static Function
RunReport(cTitulo,cString,cNomeProg,cTamanho,nTipo,nLimite)

Local nLin          := 80
Local cCabec1      := ""
Local cCabec2      := ""
Local cArqInd

cCabec1 := "CÓDIGO"+Space(2)+"LOJA"+Space(2)+"NOME
REDUZIDO"+Space(9)
cCabec1 += "RAZÃO
SOCIAL"+Space(30)+"CNPJ"+Space(18)+"INSCR.ESTADUAL"+Space(8)
cCabec1 += "CEP"
```

```

cCabec2 := "ESTADO"+Space(2)+"MUNICIPIO"+Space(8)+"ENDERECO"

dbSelectArea("TRBSA1")
dbGoTop()

SetRegua(RecCount())

While !EOF()

    If lAbortPrint .OR. nLastKey == 27
        @nLin,00 PSAY "*** CANCELADO PELO OPERADOR ***"
        Exit
    Endif

    If nLin > 55 // Salto de Página. Neste caso o formulario tem 55
linhas...
        Cabec(cTitulo,cCabec1,cCabec2,cNomeProg,cTamanho,nTipo)
        nLin := 9
    Endif

    // Primeira linha de detalhe:
    @nLin,000 PSAY TRBSA1->A1_COD
    @nLin,008 PSAY TRBSA1->A1_LOJA
    @nLin,014 PSAY TRBSA1->A1_NREDUZ
    @nLin,036 PSAY TRBSA1->A1_NOME
    @nLin,078 PSAY TRBSA1->A1_CGC
    @nLin,100 PSAY TRBSA1->A1_INSCR
    @nLin,122 PSAY TRBSA1->A1_CEP
    nLin++

    // Segunda linha de detalhe
    @nLin,000 PSAY TRBSA1->A1_EST
    @nLin,008 PSAY TRBSA1->A1_MUN
    @nLin,025 PSAY TRBSA1->A1_END
    nLin++

    //Linha separadora de detalhes
    @nLin,000 PSAY Replicate("-",nLmite)
    nLin++

    dbSkip() // Avanca o ponteiro do registro no arquivo
EndDo

SET DEVICE TO SCREEN

If aReturn[5]==1
    dbCommitAll()
    SET PRINTER TO
    OurSpool(wnrel)

```

```
Endif  
  
MS_FLUSH()  
RETURN
```

## OURSPOOL()

---

A função OURSPOOL() executa o gerenciador de impressão da aplicação Protheus, permitindo a visualização do arquivo de impressão gerado pelo relatório no formato PostScrip® com extensão “##R”.

**Sintaxe:** OURSPOOL(cArquivo)

**Parâmetros:**

cArquivo	Nome do relatório a ser visualizado.
----------	--------------------------------------

**Retorno:**

Nenhum	()
--------	----

**Exemplo:**

```
If aReturn[5]==1 // Indica impressão em disco.  
    dbCommitAll()  
    SET PRINTER TO  
    OurSpool(wnrel)  
Endif
```

## RODA()

---

A função RODA() imprime o rodapé da página do relatório, o que pode ser feito a cada página, ou somente ao final da impressão.



Dica

Pode ser utilizado o ponto de entrada "RodaEsp" para tratamento de uma impressão específica.

**Sintaxe:** Roda(uPar01, uPar02, cSize)

**Parâmetros:**

<b>uPar01</b>	Não é mais utilizado.
<b>uPar02</b>	Não é mais utilizado.
<b>cSize</b>	Tamanho do relatório ("P","M","G").

**Retorno:**

<b>Nenhum</b>	()
---------------	----

**Exemplo:**

```
/*
+-----
| Função      | TESTIMPR     | Autor | MICROSIGA           | Data |
01.01.2007 |
+-----
| Descrição      | Exemplo de utilização da função RODA() em
conjunto com a CABEC.|
|+-----
| Uso          | Curso ADVPL
|+-----
*/
#include "protheus.ch"

User Function TestImpr()
Local wnrel
Local cString := "SA1"
Local titulo := "Teste Impressão de Relatorios"
Local NomeProg := "XXX"
Local Tamanho := "M"

PRIVATE aReturn := { "Zebrado", 1,"Administracao", 1, 2, 1, "",1 }

wnrel:=SetPrint(cString,NomeProg,"",@titulo,"", "",
",,F,,F,,F,,Tamanho,,F,)

SetDefault(aReturn,cString)

RptStatus({|IEnd|
TestRel(@IEnd,wnRel,cString,Tamanho,NomeProg)},titulo)

Return
```

**Exemplo (continuação):**

```
/*
+-----
| Função      | TESTREL      | Autor | MICROSIGA          | Data |
01.01.2007 |
+-----
| Descrição      | Função interna de impressão da TestImpr().      |
|+-----|
| Uso          | Curso ADVPL      |
|+-----*/
User Function TestRel(lEnd,WnRel,cString,Tamanho,NomeProg)

LOCAL cabec1,cabec2
LOCAL cRodaTxt := oemtoansi("Rodapé")
Local nCntImpr
Local nTipo

nCntImpr := 0
li := 80
m_pag := 1
nTipo := 15
titulo:= oemtoansi("Lista de Clientes")
cabec1:= oemtoansi("COD LOJA NOME"+Space(27)+ "NOME FANTASIA")
cabec2:=""

dbSelectArea("SA1")
dbGoTop()
SetRegua(LastRec())
While !Eof()
    IncRegua()
    If Li > 60
        cabec(titulo,cabec1,cabec2,nomeprog,tamanho,15)
        @ Li,0 PSAY __PrtThinLine()
    Endif
    nCntImpr++
    Li++
    @ Li,01 PSAY A1_COD
    @ Li,05 PSAY A1_LOJA
    @ Li,10 PSAY A1_NOME
    @ Li,51 PSAY A1_NREDUZ
    If Li > 60
        Li:=66
    Endif
    dbSkip()
EndDO

If li != 80
```

```

    Roda(nCntImpr,cRodaTxt,Tamanho)
EndIf

Set Device to Screen
If aReturn[5] = 1
    Set Printer To
    dbCommitAll()
    OurSpool(wnrel)
Endif
MS_FLUSH()
Return

```

## **SETDEFAULT()**

---

A função SetDefault() prepara o ambiente de impressão de acordo com as informações configuradas no *array* aReturn, obtidas através da função SetPrint().

**Sintaxe:** **SetDefault ( < aReturn > , < cAlias > , [ uParm3 ] , [ uParm4 ] ,
[cSize] , [ nFormat ] )**

**Parâmetros:**

<b>aReturn</b>	Configurações de impressão.
<b>cAlias</b>	Alias do arquivo a ser impresso.
<b>uParm3</b>	Parâmetro reservado.
<b>uParm4</b>	Parâmetro reservado.
<b>cSize</b>	Tamanho da página "P","M" ou "G"
<b>nFormat</b>	Formato da página, 1 retrato e 2 paisagem.

**Retorno:**

<b>Nenhum</b>	()
---------------	----

---

#### Estrutura aReturn:

<b>aReturn[1]</b>	Caracter, tipo do formulário.
<b>aReturn[2]</b>	Numérico, opção de margem.
<b>aReturn[3]</b>	Caracter, destinatário.
<b>aReturn[4]</b>	Numérico, formato da impressão.
<b>aReturn[5]</b>	Numérico, dispositivo de impressão.
<b>aReturn[6]</b>	Caracter, driver do dispositivo de impressão.
<b>aReturn[7]</b>	Caracter, filtro definido pelo usuário.
<b>aReturn[8]</b>	Numérico, ordem.
<b>aReturn[x]</b>	A partir a posição [9] devem ser informados os nomes dos campos que devem ser considerados no processamento, definidos pelo uso da opção Dicionário da SetPrint().

#### **SETPRC()**

---

A função SETPRC() é utilizada para posicionar o dispositivo de impressão ativo, previamente definido pelo uso das funções AVALIMP() ou SETPRINT(), em uma linha/coluna especificada.

##### Sintaxe: SETPRC(**nLinha, nColuna**)

##### Parâmetros:

<b>nLinha</b>	Linha na qual deverá ser posicionado o dispositivo de impressão.
<b>nColuna</b>	Coluna na qual deverá ser posicionado o dispositivo de impressão.

##### Retorno:

<b>Nenhum</b>	()
---------------	----

#### Exemplo:

```
aReturn := { "", 1, "", 2, 3, cPorta , "",IndexOrd() }
SetPrint(Alias(),"","","","","F.,,,,'EPSON.DRV',.T.,,cPorta)
if nLastKey == 27
Return (.F.)
Endif
SetDefault(aReturn,Alias())
SetPrc(0,0)
```

## **SETPRINT()**

---

A função SetPrint() cria uma interface padrão onde as opções de impressão de um relatório podem ser configuradas. Basicamente duas variáveis m\_pag e aReturn precisam ser declaradas como privadas (private) antes de executar a SetPrint(), sendo que:

- ❑ **m\_pag**: controla o número de páginas.
- ❑ **aReturn**: vetor contendo as opções de impressão, sendo sua estrutura básica composta de 8 (oito) elementos.

Após confirmada, os dados são armazenados no vetor aReturn que será passado como parâmetro para função SetDefault().

**Sintaxe:** **SetPrint ( < cAlias > , < cProgram > , [ cPergunte ] , [ cTitle ] , [ cDesc1 ] , [ cDesc2 ] , [ cDesc3 ] , [ IDic ] , [ aOrd ] , [ ICompres ] , [ cSize ] , [ uParm12 ] , [ IFilter ] , [ ICrystal ] , [ cNameDrv ] , [ uParm16 ] , [ IServer ] , [ cPortPrint ] ) --> cReturn**

**Parâmetros:**

<b>cAlias</b>	Alias do arquivo a ser impresso.
<b>cProgram</b>	Nome do arquivo a ser gerado em disco.
<b>cPergunte</b>	Grupo de perguntas cadastrado no dicionário SX1.
<b>cTitle</b>	Título do relatório.

**Parâmetros (continuação):**

<b>cDesc1</b>	Descrição do relatório.
<b>cDesc2</b>	Continuação da descrição do relatório.
<b>cDesc3</b>	Continuação da descrição do relatório.
<b>IDic</b>	Utilizado na impressão de cadastro genérico permite a escolha dos campos a serem impressos. Se o parâmetro cAlias não for informado o valor padrão assumido será .F.
<b>aOrd</b>	Ordem(s) de impressão.
<b>ICompres</b>	Se verdadeiro (.T.) permite escolher o formato da impressão, o valor padrão assumido será .T.
<b>cSize</b>	Tamanho do relatório "P","M" ou "G".
<b>uParm12</b>	Parâmetro reservado.
<b>IFilter</b>	Se verdadeiro (.T.) permite a utilização do assistente de filtro, o valor padrão assumido será .T.
<b>ICrystal</b>	Se verdadeiro (.T.) permite integração com Crystal Reports, o valor padrão assumido será .F.
<b>cNameDrv</b>	Nome de um driver de impressão.

<b>uParm16</b>	Parâmetro reservado.
<b>IServer</b>	Se verdadeiro (.T.) força impressão no servidor.
<b>cPortPrint</b>	Define uma porta de impressão padrão.

**Retorno:**

<b>Caracter</b>	Nome do Relatório
-----------------	-------------------

**Estrutura aReturn:**

<b>aReturn[1]</b>	Caracter, tipo do formulário.
<b>aReturn[2]</b>	Numérico, opção de margem.
<b>aReturn[3]</b>	Caracter, destinatário.
<b>aReturn[4]</b>	Numérico, formato da impressão.
<b>aReturn[5]</b>	Numérico, dispositivo de impressão.
<b>aReturn[6]</b>	Caracter, driver do dispositivo de impressão.
<b>aReturn[7]</b>	Caracter, filtro definido pelo usuário.
<b>aReturn[8]</b>	Numérico, ordem.
<b>aReturn[x]</b>	A partir a posição [9] devem ser informados os nomes dos campos que devem ser considerados no processamento, definidos pelo uso da opção Dicionário da SetPrint().

## Controle de processamentos

---

### **ABREEXCL()**

A função ABREEXCL() fecha o arquivo cujo alias está expresso em <cAlias> e o reabre em modo exclusivo para proceder operações em que isto é necessário, como por exemplo, PACK. Se outra estação estiver usando o arquivo, o retorno será .F..

**Sintaxe: ABREEXCL(cAlias)**

**Parâmetros:**

<b>cAlias</b>	Alias do arquivo que será reaberto em modo exclusivo.
---------------	---

**Retorno:**

<b>Lógico</b>	Indica se foi possível abrir o arquivo em modo exclusivo.
---------------	---

## **CLOSEOPEN()**

---

A função CLOSEOPEN() é utilizada para fechar e reabrir uma lista de arquivos especificada.

**Sintaxe:** **CLOSEOPEN(aClose, aOpen)**

**Parâmetros:**

<b>aClose</b>	<i>Array</i> contendo os Aliases dos arquivos que deverão ser fechados.
<b>aOpen</b>	<i>Array</i> contendo os Aliases dos arquivos que deverão ser abertos.

**Retorno:**

<b>Lógico</b>	Indica se todos os arquivos especificados em aOpen foram abertos com sucesso.
---------------	---

## **CLOSESFILE()**

---

A função CLOSESFILE() fecha todos os arquivos em uso pela conexão, com exceção dos SXs (inclusive SIX), SM2 e SM4.

**Sintaxe:** **CLOSESFILE(cAlias)**

**Parâmetros:**

<b>cAlias</b>	<i>String</i> contendo os nomes dos demais Aliases que não deverão ser fechados, separando os itens com “/”.
---------------	--

**Retorno:**

<b>Lógico</b>	Indica se todos os arquivos foram fechados com sucesso.
---------------	---

## **CHKFILE()**

---

A função CHKFILE() retorna verdadeiro (.T.) se o arquivo já estiver aberto ou se o Alias não for informado. Sempre que desejar mudar o modo de acesso do arquivo (de compartilhado para exclusivo ou vice-versa), feche-o antes de chamá-la.

**Sintaxe:** **ChkFile(cAlias,lExcl,cNewAlias)**

**Parâmetros:**

cAlias	Alias do arquivo a ser reaberto.
lExcl	Se for informado verdadeiro (.T.), o arquivo será aberto em modo Exclusivo. Caso contrário, o arquivo será aberto em modo compartilhado. Se este parâmetro não for informado, será assumido falso (.F.).
cNewAlias	Novo Alias para reabertura do arquivo.

**Retorno:**

Lógico	Indica se o arquivo foi re-aberto com sucesso.
--------	--

**Exemplo:**

```
dbSelectArea("SA1")
dbCloseArea()
lOk := .T.
While .T.
IF !ChkFile("SA1",.T.)
    nResp := Alert("Outro usuário usando! Tenta de
novo?", {"Sim", "Nao"})
    If nResp == 2
        lOk := .F.
        Exit
    Endif
Endif
EndDo
If lOk
    // Faz o processamento com o arquivo...
Endif
// Finaliza
If Select("SA1")
    dbCloseArea()
Endif
ChkFile("SA1",.F.)
Return
```

## **CONOUT()**

---

A função CONOUT() permite a exibição de uma mensagem de texto no console do Server do Protheus. Caso o Protheus esteja configurado como serviço a mensagem será gravada no arquivo de log.

**Sintaxe:** CONOUT(cMensagem)

**Parâmetros:**

<b>cMensagem</b>	<i>String</i> contendo a mensagem que deverá ser exibida no console do Protheus.
------------------	--

**Retorno:**

<b>Nenhum</b>	()
---------------	----

## **CRIAVAR()**

---

A função CRIAVAR() cria uma variável, retornando o valor do campo, de acordo com o dicionário de dados, inclusive avaliando o inicializador padrão, permitindo um retorno de acordo com o tipo de dado definido no dicionário.

**Sintaxe:** CriaVar(cCampo,lIniPad,cLado)

**Parâmetros:**

<b>cCampo</b>	Nome do campo.
<b>lIniPad</b>	Indica se considera (.T.) ou não (.F.) o inicializador.
<b>cLado</b>	Se a variável for caracter, cLado pode ser: "C" - centralizado, "L" - esquerdo ou "R" – direito.

**Retorno:**

<b>Indefinido</b>	Tipo de dado de acordo com o dicionário de dados, considerando inicializador padrão
-------------------	---

**Exemplo:**

```
// Exemplo do uso da função CriaVar:  
cNumNota := CriaVar("F2_DOC") // Retorna o conteúdo do  
// inicializador padrão,  
// se existir, ou espaços em branco  
Alert(cNumNota)  
Return
```

## **DISARMTRANSACTION()**

---

A função DISARMTRANSACTION() é utilizada para realizar um “Rol Back” de uma transação aberta com o comando BEGIN TRANSACTION e delimitada com o comando END TRANSACTION.

Ao utilizar esta função, todas as alterações realizadas no intervalo delimitado pela transação são desfeitas, restaurando a situação da base de dados ao ponto imediatamente anterior ao início do processamento.

---



O uso da função DISARMTRANSACTION() não finaliza a conexão ou o processamento corrente.

**Importante**

Caso seja necessário além de desfazer as alterações, finalizar o processamento, deverá ser utilizada a função USEREXCEPTION().

---

### **Sintaxe: DISARMTRANSACTION()**

#### **Parâmetros:**

<b>Nenhum</b>	()
---------------	----

#### **Retorno:**

<b>Nenhum</b>	()
---------------	----

#### **Exemplo:**

```
IMsErroAuto := .F.  
MSExecAuto({{|x,y| MATA240(x,y)}}, aCampos, 3)  
  
If IMsErroAuto  
  
    aAutoErro := GETAUTOGRLLOG()  
    DisarmTransaction()  
    MostraErro()  
  
EndIf
```

## **EXECBLOCK()**

---

A função EXECBLOCK() executa uma função de usuário que esteja compilada no repositório. Esta função é normalmente utilizada pelas rotinas padrões da aplicação Protheus para executar pontos de entrada durante seu processamento.

---



**Importante**

A função de usuário executada através da EXECBLOCK() não recebe parâmetros diretamente, sendo que estes estarão disponíveis em uma variável private denominada **PARAMIXB**.

---



**Importante**

A variável **PARAMIXB** é o reflexo do parâmetro xParam, definido na execução da função EXECBLOCK(). Caso seja necessária a passagem de várias informações, as mesmas deverão ser definidas na forma de um *array*, tornando **PARAMIXB** um *array* também, a ser tratado na função de usuário que será executada.

---

## **EXISTBLOCK()**

---

A função EXISTBLOCK() verifica a existência de uma função de usuário compilada no repositório de objetos da aplicação Protheus. Esta função é normalmente utilizada nas rotinas padrões da aplicação Protheus para determinar a existência de um ponto de entrada e permitir sua posterior execução.

**Sintaxe:** EXISTBLOCK(**cFunção**)

**Parâmetros:**

<b>cFunção</b>	Nome da função que será avaliada.
----------------	-----------------------------------

**Retorno:**

<b>Lógico</b>	Indica se a função de usuário existe compilada no repositório de objetos corrente.
---------------	--

**Exemplo:**

```
IF EXISTBLOCK("MT100GRV")
    EXECBLOCK("MT100GRV",.F.,.F.,aParam)
ENDIF
```

## Sintaxe: EXECBLOCK(cFunção, lReserv1, lReserv2, xParam)

### Parâmetros:

cFunção	Nome da função de usuário que será executada.
lReserv1	Parâmetro de uso reservado da aplicação. Definir como .F.
lReserv2	Parâmetro de uso reservado da aplicação. Definir como .F.
xParam	Conteúdo que ficará disponível na função de usuário executada, na forma da variável private PARAMIXB.

### Retorno:

Indefinido	O retorno da EXECBLOCK() é definido pela função que será executada.
------------	---

### Exemplo:

```
aParam := {cNota, cSerie, cFornece, cLoja}

IF EXISTBLOCK("MT100GRV")
    lGravou := EXECBLOCK("MT100GRV",.F.,.F.,aParam)
ENDIF

USER FUNCTION MT100GRV()

LOCAL cNota := PARAMIXB[1]
LOCAL cSerie:= PARAMIXB[1]
LOCAL cFornece:= PARAMIXB[1]
LOCAL cLoja:= PARAMIXB[1]

RETURN .T.
```

## ERRORBLOCK()

A função ERRORBLOCK() efetua o tratamento de erros e define a atuação de um *handler* de erros sempre que ocorrer um erro em tempo de execução. O manipulador de erros é especificado como um bloco de código da seguinte forma:

{ |<objError>| <lista de expressões>, ... }, onde:

<objError> é um *error object* que contém informações sobre o erro. Dentro do bloco de código, podem ser enviadas mensagens ao *error object* para obter informações sobre o erro. Se o bloco de tratamento de erros retornar verdadeiro (.T.), a operação que falhou é repetida, e se retornar falso (.F.), o processamento recomeça.

Se não foi especificado nenhum <bErrorHandler> utilizando ERRORBLOCK() e ocorrer um erro em tempo de execução, o bloco de tratamento ao de erros padrão é avaliado. Este manipulador de erros exibe uma mensagem descritiva na tela, ajusta a função ERRORLEVEL() para 1, e depois sai do programa (QUIT).

Como ERRORBLOCK() retorna o bloco de tratamento ao de erros correntes, é possível especificar um bloco de tratamento de erros para uma operação gravando-se o bloco de manipulação de erros correntes e depois recuperando-o após o final da operação. Além disso, uma importante consequência do fato de os blocos de tratamento de erros serem especificados como blocos de código, é que eles podem ser passados para rotinas e funções definidas por usuário e depois retornadas como valores.

#### Sintaxe: ERRORBLOCK ( < bErrorHandler > )

##### Parâmetros:

<b>bErrorHandler</b>	O bloco de código a ser executado toda vez que ocorrer um erro em tempo de execução. Quando avaliado, o <bErrorHandler> é passado na forma de um objeto erro como um argumento pelo sistema.
----------------------	--

##### Retorno:

<b>Code-block</b>	Retorna o bloco de código corrente que deve tratar o erro. Caso não tenha sido enviado nenhum bloco de tratamento de erro desde que o programa foi invocado, ERRORBLOCK() retorna o bloco de tratamento de erro padrão.
-------------------	---

##### Exemplo:

```
Function CA010Form()
LOCAL xResult
LOCAL cForm:= Upper(&(ReadVar()))
LOCAL bBlock:= ErrorBlock( { |e| ChecErro(e) } )
LOCAL cOutMod
Local IOptimize := GetNewPar("MV_OPTNFE",.F.) .Or.
GetNewPar("MV_OPTNFS",.F.)

PRIVATE IRet:=.T.

cVarOutMod := If(Type("cVarOutMod") = "U", "", cVarOutMod)
cOutMod     := cVarOutMod + If(Right(cVarOutMod, 1) = ",", "", ",")

While ! Empty(cOutMod)
    If Left(cOutMod, At(",", cOutMod) - 1) $ Upper(cForm) // 
no modulo
        Help( " ",1,"ERR_FORM,,"Variavel nao disponivel para este
modulo"
    Return .F.
```

```

        Endif
        cOutMod := Subs(cOutMod, At(";", cOutMod) + 1)
    EndDo
    If ("LERSTR$cForm .or. "LERVAL$cForm .or. "LERDATA$cForm) .And.
    M->I5_CODIGO > "499"
        Help( " ",1,"CA010TXT")
        ErrorBlock(bBlock)
        Return .F.
    Endif
    BEGIN SEQUENCE
        If !"EXECBLOCK$cForm .and. !"LERSTR$cForm .And.;           // nao
        executa execblock
            !"LERVAL$cForm .And.; // nem funcao de leitura
            !"LERDATA$cForm .And.;          // de texto no
        cadastramento
            IIf(!IOptimize,.T.,!"CTBANFS$cForm .And.
        !"CTBANFE$cForm)
            xResult := &cForm
        Endif
    END SEQUENCE
    ErrorBlock(bBlock)
    Return lRet

```

## **FINAL()**

---

A função FINAL() executa as operações básicas que garantem a integridade dos dados ao finalizar o sistema desmontando as transações (se houver), desbloqueando os semáforos e fechando as tabelas abertas, finalizando-o em seguida.

. **Sintaxe:** Final( [cMensagem1], [cMensagem2] )

. **Parâmetros:**

<b>cMensagem1</b>	Primeira mensagem
<b>cMensagem2</b>	Segunda mensagem

. **Retorno:**

<b>Nenhum</b>	()
---------------	----

### **Exemplo:**

```
User Function ValidUser( cUsuario, cSenha )  
  
Local cMensag1 := "Usuário invalido!"  
Local cMensag2 := "Opção disponível para usuários Adminstradores!"  
  
If !PswAdmin( cUsuario, cSenha )  
Final( cMensag1, cMensag2 )  
Endif  
  
Return
```

### **FINDFUNCTION()**

---

A função FINDFUNCTION() tem como objetivo verificar se uma determinada função se encontra no repositório de objetos e até mesmo do binário do Protheus, sendo uma função básica da linguagem.

#### **Sintaxe: FINDFUNCTION(cFunção)**

#### **Parâmetros:**

<b>cFunção</b>	Nome da função que será avaliada no repositório de objetos corrente.
----------------	--

#### **Retorno:**

<b>Lógico</b>	Indica se a função existe compilada no repositório de objetos corrente.
---------------	---

### **FUNDESC()**

---

A função FunDesc() retornará a descrição de uma opção selecionada no menu da aplicação.

#### **Sintaxe: FUNDESC()**

#### **Parâmetros:**

<b>Nenhum</b>	()
---------------	----

#### **Retorno:**

<b>Caracter</b>	Descrição da opção selecionada no menu da aplicação.
-----------------	--

## **FUNNAME()**

---

A função FunName() retornará o nome de uma função executada a partir de um menu da aplicação.

### **Sintaxe: FUNNAME()**

#### **Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

#### **Retorno:**

<b>Caracter</b>	Nome da função executada a partir do menu da aplicação.
-----------------	---

## **GETAREA()**

---

Função utilizada para proteger o ambiente ativo no momento de algum processamento específico. Para salvar uma outra área de trabalho (alias) que não o ativo, a função GetArea() deve ser executada dentro do alias: ALIAS->(GetArea()).

### **Sintaxe: GETAREA()**

#### **Retorno: Array contendo {Alias(),IndexOrd(),Recno()}**

#### **Parâmetros**

<b>Nenhum</b>	( )
---------------	-----

## **GETCOUNTRYLIST()**

---

A função GETCOUNTRYLIST() retorna um array de duas dimensões contendo informações dos países localizados.

### **Sintaxe: GetCountryList()**

#### **Parâmetros:**

<b>Nenhum</b>	( )
---------------	-----

#### **Retorno:**

<b>Array</b>	Array de duas dimensões, sendo uma linha para cada país localizado, contendo em cada posição a sigla dos países, o nome do país e a identificação do país com dois dígitos.
--------------	---

### **Exemplo:**

```

Local aArray := GetCountryList()
Local cSigla := GetMv( "MV_PAISLOC" )
Local nPos

nPos := Ascan( aArray, { |d| d[1] == Upper(cSigla) } )
If nPos > 0
  APMsgInfo( "País de localização " + aArray[nPos,2] )
EndIf

```

### **ISINCALLSTACK()**

A função ISINCALLSTACK() verifica se uma determinada função está existindo dentro da pilha de chamadas do processamento corrente.

**Sintaxe:** IsInCallStack( **clsInCallStack** , **cStackExit** )

**Parâmetros:**

<b>clsInCallStack</b>	Nome da função que desejassem pesquisar na pilha.
<b>cStackExit</b>	<i>String</i> que identifica o ponto em que desejassem finalizar a busca. Caso não seja informada, será utilizada como padrão a expressão "STACK_EXIT".

**Retorno:**

<b>Lógico</b>	Indica se a função especificada encontrasse na pilha de chamadas do processamento corrente, até o ponto de saída especificado.
---------------	--

### **REGTOMEMORY()**

Inicializa as variáveis de memória identificadas pelo uso do alias “M->” de acordo com a estrutura e/ou informações contidas no arquivo definido como referência.

**Sintaxe:** REGTOMEMORY(**cAlias**, **lInclui**)

**Parâmetros:**

<b>cAlias</b>	Alias do arquivo que será utilizado como referência para inicialização das variáveis de memória.
<b>lInclui</b>	Identifica se as variáveis deverão ser inicializadas com conteúdos padrões, ou contendo as informações do registro posicionado do alias especificado.

**Retorno:**

Nenhum	()
--------	----

**RESTAREA()**

Função utilizada para devolver a situação do ambiente salva através do comando GETAREA(). Deve-se observar que a última área restaurada é a área que ficará ativa para a aplicação.

**Sintaxe: RESTAREA(aArea)**

**Parâmetros**

aArea	Array contendo: {cAlias, nOrdem, nRecno}, normalmente gerado pelo uso da função GetArea().
-------	--

**Exemplo:**

```
// ALIAS ATIVO ANTES DA EXECUÇÃO DA ROTINA          SN3
User Function XATF001()

LOCAL cVar
LOCAL aArea := GetArea()
LOCAL IRet := .T.

cVar := &(ReadVar())

dbSelectArea("SX5")
IF !dbSeek(xFilial()+"Z1"+cVar)

    CSTR0001 := "REAV - Tipo de Reavaliacao"
    CSTR0002 := "Informe um tipo de reavaliacao valido"
    CSTR0003 := "Continuar"
    Aviso(cSTR0001,cSTR0002,{cSTR0003},2)
    IRet := .F.

ENDIF

RestArea(aArea)
Return( IRet )
```

## **USEREXCEPTION()**

---

A função USEREXCEPTION() tem o objetivo de forçar um erro em ADVPL de forma que possamos tratar de alguma forma. USEREXCEPTION() recebe uma *string* contendo uma descrição do erro, essa descrição será exibida de acordo com o ambiente que se está executando (no caso, um ambiente ERP). Será exibida uma tela de erro.

**Sintaxe:** **USEREXCEPTION(cMensagem)**

**Parâmetros:**

<b>cMensagem</b>	Mensagem que será exibida no cabeçalho do erro, contendo a explicação da exceção.
------------------	---

**Retorno:**

<b>Nenhum</b>	()
---------------	----

## **Utilização de recursos do ambiente ERP**

---

### **AJUSTASX1()**

---

A função AJUSTASX1() permite a inclusão simultânea de vários itens de perguntas para um grupo de perguntas no SX1 da empresa ativa.

**Sintaxe:** **AJUSTASX1(cPerg, aPergs)**

**Parâmetros:**

<b>cPerg</b>	Grupo de perguntas do SX1 (X1_GRUPO)
<b>aPergs</b>	Array contendo a estrutura dos campos que serão gravados no SX1.

**Retorno:**

<b>Nenhum</b>	()
---------------	----

**Estrutura – Item do array aPerg:**

Posição	Campo	Tipo	Descrição
01	X1_PERGUNT	Caractere	Descrição da pergunta em português.
02	X1_PERSPA	Caractere	Descrição da pergunta em espanhol.
03	X1_PERENG	Caractere	Descrição da pergunta em inglês.
04	X1_VARIAVL	Caractere	Nome da variável de controle auxiliar (mv_ch).
05	X1_TIPO	Caractere	Tipo do parâmetro.
06	X1_TAMANHO	Numérico	Tamanho do conteúdo do parâmetro.
07	X1_DECIMAL	Numérico	Número de decimais para conteúdos numéricos.
08	X1_PRESEL	Numérico	Define qual opção do combo é a padrão para o parâmetro.
09	X1_GSC	Caractere	Define se a pergunta será do tipo G – Get ou C – Choice (combo).
10	X1_VALID	Caractere	Expressão de validação do parâmetro.
11	X1_VAR01	Caractere	Nome da variável MV_PAR+”Ordem” do parâmetro.
12	X1_DEF01	Caractere	Descrição da opção 1 do combo em português.
13	X1_DEFSPA1	Caractere	Descrição da opção 1 do combo em espanhol.
14	X1_DEFENG1	Caractere	Descrição da opção 1 do combo em inglês.
15	X1_CNT01	Caractere	Conteúdo padrão ou último conteúdo definido como respostas para a pergunta.
16	X1_VAR02	Caractere	Não é informado.
17	X1_DEF02	Caractere	Descrição da opção X do combo em português.
18	X1_DEFSPA2	Caractere	Descrição da opção X do combo em espanhol.
19	X1_DEFENG2	Caractere	Descrição da opção X do combo em inglês.
20	X1_CNT02	Caractere	Não é informado.
21	X1_VAR03	Caractere	Não é informado.
22	X1_DEF03	Caractere	Descrição da opção X do combo em português.

**Estrutura – Item do array aPerg (continuação):**

<b>23</b>	X1_DEFSPA3	Caractere	Descrição da opção X do combo em espanhol.
<b>24</b>	X1_DEFENG3	Caractere	Descrição da opção X do combo em inglês.
<b>25</b>	X1_CNT03	Caractere	Não é informado.
<b>26</b>	X1_VAR04	Caractere	Não é informado.
<b>27</b>	X1_DEF04	Caractere	Descrição da opção X do combo em português.
<b>28</b>	X1_DEFSPA4	Caractere	Descrição da opção X do combo em espanhol.
<b>29</b>	X1_DEFENG4	Caractere	Descrição da opção X do combo em inglês.
<b>30</b>	X1_CNT04	Caractere	Não é informado.
<b>31</b>	X1_VAR05	Caractere	Não é informado.
<b>32</b>	X1_DEF05	Caractere	Descrição da opção X do combo em português.
<b>33</b>	X1_DEFSPA5	Caractere	Descrição da opção X do combo em espanhol.
<b>34</b>	X1_DEFENG5	Caractere	Descrição da opção X do combo em inglês.
<b>35</b>	X1_CNT05	Caractere	Não é informado.
<b>36</b>	X1_F3	Caractere	Código da consulta F3 vinculada ao parâmetro.
<b>37</b>	X1_GRPSXG	Caractere	Código do grupo de campos SXG para atualização automática, quando o grupo for alterado.
<b>38</b>	X1_PYME	Caractere	Se a pergunta estará disponível no ambiente Pyme.
<b>39</b>	X1_HELP	Caractere	Conteúdo do campo X1_HELP
<b>40</b>	X1_PICTURE	Caractere	Picture de formatação do conteúdo do campo.
<b>41</b>	aHelpPor	Array	Vetor simples contendo as linhas de help em português para o parâmetro. Trabalhar com linhas de até 40 caracteres.
<b>42</b>	aHelpEng	Array	Vetor simples contendo as linhas de help em inglês para o parâmetro. Trabalhar com linhas de até 40 caracteres.
<b>43</b>	aHelpSpa	Array	Vetor simples contendo as linhas de help em espanhol para o parâmetro. Trabalhar com linhas de até 40 caracteres.

## **ALLUSERS()**

---

A função ALLUSERS() retorna um *array* multidimensional contendo as informações dos usuários do sistema.

### **Sintaxe: ALLUSERS()**

#### **Parâmetros:**

<b>Nenhum</b>	<b>()</b>
---------------	-----------

#### **Retorno:**

<b>Array</b>	<i>Array multidimensional contendo as informações dos usuários do sistema,</i> aonde para cada usuário serão demonstradas as seguintes informações:  aArray[x][1] ☐ Configurações gerais de acesso aArray[x][2] ☐ Configurações de impressão aArray[x][3] ☐ Configurações de acesso aos módulos
--------------	--

#### **Array de informações dos usuários: Configurações gerais de acesso**

<b>Elemento</b>	<b>Descrição</b>	<b>Tipo</b>	<b>Ctd.</b>
<b>1</b>			
<b>1</b>	ID	C	6
<b>2</b>	Nome	C	15
<b>3</b>	Senha	C	6
<b>4</b>	Nome Completo	C	30
<b>5</b>	Vetor com nº últimas senhas	A	--
<b>6</b>	Data de validade	D	8
<b>7</b>	Quantas vezes para expirar	N	4
<b>8</b>	Autorizado a alterar a senha	L	1
<b>9</b>	Alterar a senha no próximo logon L		1
<b>10</b>	Vetor com os grupos	A	--
<b>11</b>	ID do superior	C	6
<b>12</b>	Departamento	C	30
<b>13</b>	Cargo	C	30
<b>14</b>	E-Mail	C	130
<b>15</b>	Número de acessos simultâneos	N	4
<b>16</b>	Data da última alteração	D	8
<b>17</b>	Usuário bloqueado	L	1
<b>18</b>	Número de dígitos para o ano	N	1
<b>19</b>	Listner de ligações	L	1
<b>20</b>	Ramal	C	4

### **Array de informações dos usuários: Configurações de impressão**

Elemento	Descrição		Tipo	Qtd.
<b>2</b>				
	<b>1</b>	Vetor com horários de acesso	A	--
	<b>2</b>	Idioma	N	1
	<b>3</b>	Diretório	C	100
	<b>4</b>	Impressora	C	--
	<b>5</b>	Acessos	C	512
	<b>6</b>	Vetor com empresas	A	--
	<b>7</b>	Ponto de entrada	C	10
	<b>8</b>	Tipo de impressão	N	1
	<b>9</b>	Formato	N	1
	<b>10</b>	Ambiente	N	1
	<b>11</b>	Prioridade p/ config. do grupo	L	1
	<b>12</b>	Opção de impressão	C	50
	<b>13</b>	Acesso a outros dir de impressão	L	1

### **Array de informações dos usuários: Configurações de acesso aos módulos**

Elemento	Descrição		Tipo	Qtd.
<b>3</b>				
	<b>1</b>	Módulo+nível+menu	C	

## **ALLGROUPS()**

A função ALLGROUPS() retorna um *array* multidimensional contendo as informações dos grupos de usuários do sistema.

### **Sintaxe: ALLGROUPS()**

#### **Parâmetros:**

<b>Nenhum</b>	<b>()</b>
---------------	-----------

#### **Retorno:**

<b>Array</b>	<i>Array</i> multidimensional contendo as informações dos grupos de usuários do sistema, aonde para cada grupo serão demonstradas as seguintes informações:  aArray[x][1] ☐ Configurações gerais de acesso aArray[x][2] ☐ Configurações de acesso aos módulos
--------------	--

### Array de informações dos grupos: Configurações gerais de acesso

Elemento	Descrição		Tipo	Qtd.
1				
1	ID		C	6
2	Nome		C	20
3	Vetor com horários de acesso		A	
4	Data de validade		D	8
5	Quantas vezes para expirar		N	4
6	Autorizado a alterar a senha		L	1
7	Idioma		N	1
8	Diretório		C	100
9	Impressora		C	
10	Acessos		C	512
11	Vetor com empresas		A	
12	Data da última alteração		D	8
13	Tipo de impressão		N	1
14	Formato		N	1
15	Ambiente		N	1
16	Opção de impressão		L	1
17	Acesso a outros Dir de impressão		L	1

### Array de informações dos grupos: Configurações de acesso aos módulos

Elemento	Descrição		Tipo	Qtd.
2				
	1	Modulo+nível+menu	C	

### CGC()

---

A função CGC() valida o CGC digitado, utilizando o algoritmo nacional para verificação do dígito de controle.

**Sintaxe:** CGC(cCGC)

**Parâmetros:**

cCGC	String contendo o CGC a ser validado.
------	---------------------------------------

---

**Retorno:**

Lógico	Indica se o CGC informado é válido.
--------	-------------------------------------

## **CONPAD1()**

---

A função CONPAD1() exibe uma tela de consulta padrão baseada no Dicionário de Dados (SXB).

**Sintaxe:** ConPad1 ( [uPar1], [uPar2], [uPar3], cAlias, [cCampoRet] , [uPar4],  
[IOnlyView] )

**Parâmetros:**

<b>uPar</b>	Parâmetro reservado.
<b>uPar2</b>	Parâmetro reservado.
<b>uPar3</b>	Parâmetro reservado.
<b>cAlias</b>	Consulta padrão cadastrada no Dicionário de Dados (SXB) a ser utilizada.
<b>cCampoRet</b>	Nome da variável ou campo que receberá o retorno da consulta padrão.
<b>uPar4</b>	Parâmetro Reservado.
<b>IOnlyView</b>	Indica se será somente para visualização.

**Retorno:**

<b>Nenhum</b>	()
---------------	----

## **DATAVALIDA()**

---

A função DATAVALIDA() retorna a primeira data válida a partir de uma data especificada como referência, considerando inclusive a data informada para análise.

**Sintaxe:** DATAVALIDA(dData)

**Parâmetros:**

<b>dData</b>	Data a partir da qual será avaliada a próxima data válida, considerando-a inclusive como uma possibilidade.
--------------	---

**Retorno:**

<b>Data</b>	Próxima data válida, desconsiderando sábados, domingos e os feriados cadastrados no sistema.
-------------	--

## **EXISTINI()**

---

A função EXISTINI() verifica se o campo possui inicializador padrão.

**Sintaxe:** EXISTINI(cCampo)

**Parâmetros:**

<b>cCampo</b>	Nome do campo para verificação.
---------------	---------------------------------

**Retorno:**

<b>Lógico</b>	Indica se o campo possui um inicializador padrão.
---------------	---

**Exemplo:**

```
// Exemplo de uso da função ExistIni:  
// Se existir inicializador no campo B1_COD:  
If ExistIni("B1_COD")  
// Executa o inicializador:  
cCod := CriaVar("B1_COD")  
Endif  
  
Return
```

## **EXTENSO()**

---

A função EXTENSO() retorna uma *string* referente à descrição por extenso de um valor numérico, sendo comumente utilizada para impressão de cheques, valor de duplicatas, etc.

**Sintaxe:** Extenso(nValor, lQtd, nMoeda)

**Parâmetros:**

<b>nValor</b>	Valor para geração do extenso.
<b>lQtd</b>	Indica se o valor representa uma quantidade (.T.) ou dinheiro (.F.).
<b>nMoeda</b>	Para qual moeda do sistema deve ser o extenso.

**Retorno:**

<b>String</b>	Descrição do valor por extenso.
---------------	---------------------------------

## **FORMULA()**

---

Interpreta uma fórmula cadastrada. Esta função interpreta uma fórmula, previamente cadastrada no Arquivo SM4 através do Módulo Configurador, e retorna o resultado com tipo de dado de acordo com a própria fórmula.

### **Sintaxe: Formula(cFormula)**

#### **Parâmetros:**

<b>cFormula</b>	Código da fórmula a ser avaliada e cadastrada no SM4 – Cadastro de Fórmulas.
-----------------	--

#### **Retorno:**

<b>Indefinido</b>	Resultado da interpretação da fórmula cadastrada no SM4.
-------------------	--

## **GETADVVAL()**

---

A função GETADVVAL() executa uma pesquisa em um arquivo pela chave de busca e na ordem especificada, possibilitando o retorno de um ou mais campos.

### **Sintaxe: GetAdvFVal(cAlias,uCpo,uChave,nOrder,uDef)**

#### **Parâmetros:**

<b>cAlias</b>	Alias do arquivo.
<b>uCpo</b>	Nome de um campo ou <i>array</i> contendo os nomes dos campos Desejados.
<b>uChave</b>	Chave para a pesquisa.
<b>nOrder</b>	Ordem do índice para a pesquisa.
<b>uDef</b>	Valor ou <i>array</i> “default” para ser retornado caso a chave não seja encontrada

#### **Retorno:**

<b>Indefinido</b>	Retorna o conteúdo de um campo ou <i>array</i> com o conteúdo de vários campos.
-------------------	---



**Importante**

A função GETADVVAL() difere da função POSICIONE() apenas por permitir o retorno de vários campos em uma única consulta.

As duas funções devem ser protegidas por GETAREA() / RESTAREA() dependendo da aplicação.

## **HELP()**

---

Esta função exibe a ajuda especificada para o campo e permite sua edição. Se for um help novo, escreve-se o texto em tempo de execução.

**Sintaxe:** Help(**cHelp**,**nLinha**, **cTitulo**, **uPar4**,**cMensagem**,**nLinMen**,**nColMen**)

**Parâmetros:**

<b>cHelp</b>	Nome da Rotina chamadora do help. (sempre branco)
<b>nLinha</b>	Número da linha da rotina chamadora. (sempre 1)
<b>cTitulo</b>	Título do help
<b>uPar4</b>	Sempre NIL
<b>cMensagem</b>	Mensagem a ser exibida para o Help.
<b>nLinMen</b>	Número de linhas da Mensagem. (relativa à janela)
<b>nColMen</b>	Número de colunas da Mensagem. (relativa à janela)

**Retorno:**

**Nenhum** ()

---



**Importante**

A função HELP() é tratada na execução das rotinas com o recurso de MSEXECAUTO(), permitindo a captura e exibição da mensagem no *log* de processamento.

Por esta razão, em rotinas que podem ser chamadas através da função MSEXECAUTO() deve-se sempre utilizar avisos utilizando esta função, para que este tipo de processamento não seja travado indevidamente.

---

**Exemplo:**

```
IF lAuto // Se for rotina automática
Help("ESPECIFICO",1,"HELP","PROCESSAMENTO","Parâmetros do JOB
Inválidos",1,0)
ELSE
    MsgAlert("Parâmetros do JOB Inválidos", "PROCESSAMENTO")
ENDIF
```

## **MESEXTENSO()**

---

A função MESEXTENSO() retorna o nome de um mês por extenso.

**Sintaxe: MESEXTENSO(nMes)**

**Parâmetros:**

<b>nMes</b>	Indica o número do mês a ter seu nome escrito por extenso.
-------------	--



*Importante*

Este parâmetro pode ser definido também como caractere ou como data.

**Retorno:**

<b>String</b>	Nome do mês indicado por extenso.
---------------	-----------------------------------

## **OBRIGATORIO()**

---

A função OBRIGATORIO() avalia se todos os campos obrigatórios de uma Enchoice() foram digitados.

**Sintaxe: OBRIGATORIO(aGets, aTela, aTitulos)**

**Parâmetros:**

<b>aGets</b>	Variável PRIVATE tratada pelo objeto Enchoice(), previamente definida no fonte.
<b>aTela</b>	Variável PRIVATE tratada pelo objeto Enchoice(), previamente definida no fonte.
<b>aTitulos</b>	Array contendo os títulos dos campos exibidos na Enchoice().

**Retorno:**

<b>Lógico</b>	Indica se todos os campos obrigatórios foram preenchidos.
---------------	---

**Exemplo:**

```
#INCLUDE "protheus.ch"
/*
+-----
| Programa | ATFA010A | Autor | ARNALDO R. JUNIOR      | Data |
+-----
| Desc.      | Cadastro de dados complementares do bem – Ativo Fixo | 
+-----
| Uso        | Curso de ADVPL
+-----
*/
User Function ATFA010A()

Private cCadastro := "Atualizacao de dados do bem"
Private aRotina := { {"Pesquisar", "AxPesqui" ,0,1} ;;
                    {"Visualizar" , "AxVisual" ,0,2} ;;
                    {"Atualizar" , "U_A010AATU" ,0,4}}}

Private cDelFunc := ".T."
Private cString := "SN1"

dbSelectArea("SN1")
dbSetOrder(1)
dbSelectArea(cString)
mBrowse( 6,1,22,75,cString)

Return

/*
+-----
| Programa | A010AATU | Autor | ARNALDO R. JUNIOR      | Data |
+-----
| Desc.      | Atualização de dados do bem – Ativo Fixo
+-----
| Uso        | Curso de ADVPL
+-----
*/
User Function A010AATU(cAlias,nReg,nOpc)

Local aCpoEnch          := {}
Local aAlter              := {}

Local aButtons            := {}
Local cAliasE             := cAlias
Local aAlterEnch         := {}
Local aPos                := {015,000,400,600}
```

```

Local nModelo          := 
Local IF3              := .F.
Local IMemoria         := .T.
Local IColumn          := .F.
Local caTela            := ""
Local INoFolder         := .F.
Local IProperty         := .F.

Private oDlg
Private oGetD
Private oEnch

Private aTEL[0][0]    // Variáveis que serão atualizadas pela
Enchoice()
Private aGETS[0]        // e utilizadas pela função OBRIGATORIO()

DbSelectArea("SX3")
DbSetOrder(1)
DbSeek(cAliasE)
//+-----+
//| Campos da enchoice
//+-----+
While !Eof() .And. SX3->X3_ARQUIVO == cAliasE
    If !(SX3->X3_CAMPO $ "A1_FILIAL") .And. cNivel >= SX3->X3_NIVEL
    .And. X3Uso(SX3->X3_USADO)
        AAdd(aCpoEnch,SX3->X3_CAMPO)
        EndIf
        DbSkip()
    End
//+-----+
//| Campos alteráveis da enchoice
//+-----+
AADD(aAlterEnch,"N1_TIPOADT")           // Controle de Adiantamentos
AADD(aAlterEnch,"N1_DESCRIC")           // Descrição
AADD(aAlterEnch,"N1_CHAPA")             // Numero da placa
AADD(aAlterEnch,"N1_FORNEC")// Fornecedor
AADD(aAlterEnch,"N1_LOJA")               // Loja do Fornecedor
AADD(aAlterEnch,"N1_NSERIE")// Serie da Nota
AADD(aAlterEnch,"N1_NFISCAL")           // Numero da Nota
AADD(aAlterEnch,"N1_NFITEM")// Item da Nota
AADD(aAlterEnch,"N1_UM")                 // Unidade de Medida
AADD(aAlterEnch,"N1_PRODUTO")           // Código do Produto
AADD(aAlterEnch,"N1_PEDIDO")// Código do Pedido de Compras
AADD(aAlterEnch,"N1_ITEMPED")           // Item do Pedido de Compras
AADD(aAlterEnch,"N1_PRCIMP")// Código do Processo de Importação
AADD(aAlterEnch,"N1_CODPAIS")           // Código do País
AADD(aAlterEnch,"N1_ORIGCPR")           // Origem de Compras
AADD(aAlterEnch,"N1_CODSP")              // Código da SP Interna
AADD(aAlterEnch,"N1_CHASSIS")           // Numero de serie

```

```

//+-----+
//| Montagem do DIALOG
//+-----+
DEFINE MSDIALOG oDlg TITLE cCadastro FROM 000,000 TO 400,600 PIXEL
    RegToMemory("SN1", .F.)

    oEnch :=      MsMGet():New(cAliasE, nReg, nOpc, /*aCRA*/,
/*cLetra*/;;
                    /*cTexto*/, aCpoEnch,aPos,aAlterEnch, nModelo,
/*nColMens*/;;
                    /*cMensagem*/, /*cTudoOk*/, oDlg, IF3, IMemoria,
IColumn;;
                    caTela, INoFolder, IProperty)

ACTIVATE MSDIALOG oDlg CENTERED;
ON INIT EnchoiceBar(oDlg,
    {|| IIF(A010AGR(aCpoEnch,aAlterEnch,nOpc),;
        oDlg:End(),.F.)}; // Botão OK
    {|| |oDlg:End(),,aButtons} // Botão Cancelar

RETURN

*/
+-----+
| Programa | A010AGR | Autor | ARNALDO R. JUNIOR | Data |
+-----+
| Desc.     | Validação da enchoice e gravação dos dados do bem |
+-----+
| Uso       | Curso de ADVPL |
+-----+
*/
Static Function A010AGR(aCpos,aAlter,nOpc)

Local aArea          := GetArea()
Local nX             := 0

IF !Obrigatorio(aGets,aTela) /*Valida o cabecalho*/
    Return .F.
ENDIF

// Atualizacao dos campos passiveis de alteracao no SN1
RecLock("SN1",.F.)
For nX := 1 to Len(aAlter)
    SN1->&(aAlter[nX]) := M->&(aAlter[nX])
Next nX
MsUnLock()

Return .T.

```

**OPENFILE()**

---

A função OPENFILE() exibe o diagnóstico de arquivos, verificando a existência dos arquivos de dados e os índices do sistema, criando-os, caso não existam. Além de abrir os arquivos de acordo com o módulo onde é executada ou de acordo com a parametrização.

**Sintaxe: OPENFILE(cEmp)**

**Parâmetros:**

<b>cEmp</b>	Empresa cujo os arquivos serão re-abertos.
-------------	--

**Retorno:**

<b>Nenhum</b>	()
---------------	----

---

**PERGUNTE()**

A função PERGUNTE() inicializa as variáveis de pergunta (mv\_par01,...) baseada na pergunta cadastrado no Dicionário de Dados (SX1). Se o parâmetro lAsk não for especificado ou for verdadeiro será exibida a tela para edição da pergunta e se o usuário confirmar as variáveis serão atualizadas e a pergunta no SX1 também será atualizada.

**Sintaxe: Pergunte( cPergunta , [lAsk] , [cTitle] )**

**Parâmetros:**

<b>cPergunta</b>	Pergunta cadastrada no Dicionário de Dados ( SX1) a ser utilizada.
<b>/Ask</b>	Indica se exibirá a tela para edição.
<b>cTitle</b>	Título do diálogo.

**Retorno:**

<b>Lógico</b>	Indica se a tela de visualização das perguntas foi confirmada (.T.) ou cancelada (.F.)
---------------	--

## PESQPICT()

---

A função PESQPICT() retorna a picture definida para um campo especificado no Dicionário de Dados (SX3).

**Sintaxe:** PesqPict(cAlias,cCampo,nTam)

**Parâmetros:**

cAlias	Alias do arquivo.
cCampo	Nome do campo.
nTam	Opcional, para campos numéricos, será usado como o tamanho do campo para definição da <i>picture</i> . Se não informado, é usado o tamanho padrão no Dicionário de Dados.

**Retorno:**

String	Picture do campo especificado.
--------	--------------------------------

## PESQPICTQT()

---

A função PESQPICTQT() retorna a *picture* de um campo numérico referente a uma quantidade, de acordo com o Dicionário de Dados (SX3). Esta função geralmente é utilizada quando há pouco espaço disponível para impressão de valores em relatórios, quando o valor nEdição não é informado, ela tem o comportamento semelhante ao da função “X3Picture”, pois busca o tamanho do campo no dicionário de dados.

**Sintaxe:** PesqPictQt(cCampo,nEdição)

**Parâmetros:**

cCampo	Nome do campo a verificar a <i>picture</i> .
nEdição	Espaço disponível para edição.

**Retorno:**

String	Picture ideal para o espaço definido por nEdição, sem a separação dos milhares por vírgula.
--------	---

## **POSICIONE()**

---

A função POSICIONE() permite o retorno do conteúdo de um campo de um registro de uma tabela especificado através de uma chave de busca.

**Sintaxe:** Posicione(cAlias, nOrdem, cChave, cCampo)

**Parâmetros:**

<b>cAlias</b>	Alias do arquivo.
<b>nOrdem</b>	Ordem utilizada.
<b>cChave</b>	Chave pesquisa.
<b>cCampo</b>	Campo a ser retornado.

**Retorno:**

<b>Indefinido</b>	Conteúdo do campo solicitado.
-------------------	-------------------------------



*Importante*

A utilização da função POSICIONE() deve ser protegida com GETAREA()  
/ RESTAREA() dependendo da aplicação.

---

## **PUTSX1()**

---

A função PUTSX1() permite a inclusão de um único item de pergunta em um grupo de definido no Dicionário de Dados (SX1). Todos os vetores contendo os textos explicativos da pergunta devem conter até 40 caracteres por linha.

**Sintaxe:** PutSx1(cGrupo, cOrdem, cPergunt, cPerSpa, cPerEng, cVar, cTipo,  
nTamanho, nDecimal, nPresel, cGSC, cValid, cF3, cGrpSxg  
,cPyme, cVar01, cDef01, cDefSpa1 , cDefEng1, cCnt01, cDef02,  
cDefSpa2, cDefEng2, cDef03, cDefSpa3, cDefEng3, cDef04,  
cDefSpa4, cDefEng4, cDef05, cDefSpa5, cDefEng5, aHelpPor,  
aHelpEng, aHelpSpa, cHelp)

**Parâmetros:**

<b>cGrupo</b>	Grupo de perguntas do SX1 (X1_GRUPO).
<b>cOrdem</b>	Ordem do parâmetro no grupo (X1_ORDEM).
<b>cPergunt</b>	Descrição da pergunta em português.
<b>cPerSpa</b>	Descrição da pergunta em espanhol.
<b>cPerEng</b>	Descrição da pergunta em inglês.
<b>cVar</b>	Nome da variável de controle auxiliar (X1_VARIAVL).
<b>cTipo</b>	Tipo do parâmetro.
<b>nTamanho</b>	Tamanho do conteúdo do parâmetro.
<b>nDecimal</b>	Número de decimais para conteúdos numéricos.
<b>nPresel</b>	Define qual opção do combo é a padrão para o parâmetro.
<b>cGSC</b>	Define se a pergunta será do tipo G – Get ou C – Choice (combo).
<b>cValid</b>	Expressão de validação do parâmetro.
<b>cF3</b>	Código da consulta F3 vinculada ao parâmetro.
<b>cGrpSxg</b>	Código do grupo de campos SXG para atualização automática, quando o grupo for alterado.
<b>cPyme</b>	Se a pergunta estará disponível no ambiente Pyme.
<b>cVar01</b>	Nome da variável MV_PAR+”Ordem” do parâmetro.
<b>cDef01</b>	Descrição da opção 1 do combo em português.
<b>cDefSpa1</b>	Descrição da opção 1 do combo em espanhol.
<b>cDefEng1</b>	Descrição da opção 1 do combo em inglês.
<b>cCnt01</b>	Conteúdo padrão ou último conteúdo definido como respostas para este item.
<b>cDef0x</b>	Descrição da opção X do combo em português.
<b>cDefSpax</b>	Descrição da opção X do combo em espanhol.
<b>cDefEngx</b>	Descrição da opção X do combo em inglês.
<b>aHelpPor</b>	Vetor simples contendo as linhas de help em português para o parâmetro.
<b>aHelpEng</b>	Vetor simples contendo as linhas de help em inglês para o parâmetro.
<b>aHelpSpa</b>	Vetor simples contendo as linhas de help em espanhol para o parâmetro.
<b>cHelp</b>	Conteúdo do campo X1_HELP.

## **RETINDEX()**

---

A função RETINDEX() restaura os índices padrões de um alias definidos no Dicionário de Dados (SIX).

### **Sintaxe: RETINDEX(cAlias)**

#### **Parâmetros:**

<b>cAlias</b>	Alias de um arquivo do sistema existente no Dicionário de Dados.
---------------	--

#### **Retorno:**

<b>Numérico</b>	Indica quantos índices padrões o alias especificado possui no Dicionário de Dados.
-----------------	--



A função RETINDEX() quando utilizada em ambientes TOP CONNECT retorna -1

*Importante*

---

## **SIXDESCRICAO()**

---

A função SIXDESCRICAO() retorna a descrição da chave de índice, de acordo com o registro posicionado no SIX e idioma corrente.

### **Sintaxe: SIXDESCRICAO()**

#### **Parâmetros:**

<b>Nenhum</b>	()
---------------	----

#### **Retorno:**

<b>String</b>	Descrição do indice posicionado no SIX de acordo com o idioma corrente.
---------------	---

#### **Exemplo:**

```
User Function <nome-da-função>( cChave, cOrdem )
Local cSixDesc := ""

dbSelectArea("SIX")
dbSetOrder(1)

If dbSeek(cChave+cOrdem)
cSixDesc := SixDescricao()
EndIf
Return
```

## **TABELA()**

---

A função TABELA() retorna o conteúdo de uma tabela cadastrada no Arquivo de Tabelas (SX5) de acordo com a chave especificada. Caso a tabela, ou a chave especificada, não exista será exibido um HELP() padrão do sistema.

**Sintaxe:** **Tabela(cTab,cChav,lPrint)**

**Parâmetros:**

<b>cTab</b>	Identificação da tabela a pesquisar (deve ser informado como caractere).
<b>cChav</b>	Chave a pesquisar na tabela informada.
<b>lPrint</b>	Indica se deve (.T.) ou não (.F.) exibir o help ou a chave NOTAB se a tabela não existir.

**Retorno:**

<b>String</b>	Conteúdo da tabela na chave especificada. Retorna nulo caso a tabela não exista ou a chave não seja encontrada.
---------------	---

## **TAMSX3()**

---

A função TAMSX3() retorna o tamanho (total e parte decimal) de um campo especificado no Dicionário de Dados (SX3).

**Sintaxe:** **TAMSX3(cCampo)**

**Parâmetros:**

<b>cCampo</b>	Nome do campo a ser consultado no Dicionário de Dados (SX3).
---------------	--

**Retorno:**

<b>Array</b>	<i>Array</i> de duas posições contendo o tamanho total e o número de decimais do campo especificado respectivamente.
--------------	--

## **TM()**

---

A função TM() retorna a picture de impressão para valores numéricos dependendo do espaço disponível.

**Sintaxe:** **TM(nValor, nEdição, nDec)**

**Parâmetros:**

<b>nValor</b>	Valor a ser avaliado.
<b>nEdição</b>	Espaço disponível para edição.
<b>nDec</b>	Número de casas decimais.

**Retorno:**

<b>String</b>	Picture ideal para edição do valor nValor.
---------------	--

---

Esta rotina leva em consideração duas variáveis:

- ❑ MV\_MILHAR – Determina se deve haver separação de milhar;
- ❑ MV\_CENT – Número de casas decimais padrão da moeda corrente.



*Importante*

Para ajustar o valor passado (nValor) ao espaço disponível (nEdição) a função verifica se pode haver separação de milhar, neste caso, a rotina eliminará tantos pontos decimais quantos sejam necessários ao ajuste do tamanho. Caso não seja possível ajustar o valor ao espaço dado, será colocado na *picture* o caracter de estouro de campo “\*”. A função também ajusta um valor ao número de decimais (nDec), sempre imprimindo a quantidade de decimais passados no parâmetro.

---

**X1DEF01()**

---

A função X1DEF01() retorna o conteúdo da primeira definição da pergunta posicionada no SX1 (caso seja combo) no idioma corrente.

**Sintaxe: X1DEF01()**

**Parâmetros:**

<b>Nenhum</b>	()
---------------	----

**Retorno:**

<b>String</b>	Conteúdo da primeira definição da pergunta no idioma corrente.
---------------	--

**Exemplo:**

```
User Function <nome-da-função>( cGrupo, cPerg )  
Local cDef01  
Local cDef02  
Local cDef03  
Local cDef04  
Local cDef05  
  
dbSelectArea("SX1")  
dbSetOrder(1)  
  
If dbSeek( cGrupo + cPerg ) // grupo da pergunta + o numero da perg.  
cDef01 := X1Def01()  
cDef02 := X1Def02()  
cDef03 := X1Def03()  
cDef04 := X1Def04()  
cDef05 := X1Def05()  
Endif  
  
Return
```

---

**X1PERGUNT()**

---

A função X1PERGUNT() retorna a descrição da pergunta posicionada no Dicionário de Dados (SX1) para o idioma corrente.

**Sintaxe: X1PERGUNT()**

**Parâmetros:**

Nenhum	()
--------	----

**Retorno:**

String	Descrição da pergunta do Dicionário de Dados (SX1) no idioma corrente.
--------	--

**Exemplo:**

```
User Function <nome-da-função>( cGrupo, cPerg )  
Local cDescr  
dbSelectArea("SX1")  
dbSetOrder(1)  
If dbSeek( cGrupo + cPerg ) // grupo da pergunta + o numero da perg.  
cDescr := X1Pergunt()  
Endif  
Return
```

## X2NOME()

---

A função X2NOME() retorna a descrição de uma tabela posicionada no Dicionário de Dados (SX2) no idioma corrente.

### Sintaxe: X2NOME()

#### Parâmetros:

Nenhum	()
--------	----

#### Retorno:

String	Descrição da tabela posicionada no Dicionário de Dados (SX2) no idioma corrente.
--------	--

#### Exemplo:

```
User Function <nome-da-função>()
Local cTabela
dbSelectArea("SX2")
dbSetOrder(1)
If dbSeek( "SA1" )
cTabela := X2Nome()
EndIf
Return
```

## X3CBOX()

---

A função X3CBOX() retorna o conteúdo de um campo tipo combo posicionado no Dicionário de Dados (SX3) no idioma corrente.

### Sintaxe: X3CBOX()

#### Parâmetros:

Nenhum	()
--------	----

#### Retorno:

String	Conteúdo do campo do tipo combo posicionado no Dicionário de Dados (SX3) no idioma corrente.
--------	--

**Exemplo:**

```
User Function <nome-da-função>( )  
  
Local cTitulo  
Local cDescri  
Local cCombo  
  
dbSelectArea("SX3")  
dbSetOrder(2)  
  
If dbSeek( cCampo )  
cTitulo := X3Titulo()  
cDescri := X3Descri()  
cCombo := X3Cbox()  
EndIf  
  
Return
```

**X3DESCRIC()**

---

A função X3DESCRIC() retorna a descrição de um campo posicionado no Dicionário de Dados (SX3) no idioma corrente.

**Sintaxe: X3DESCRIC()**

**Parâmetros:**

---

<b>Nenhum</b>	()
---------------	----

**Retorno:**

<b>String</b>	Descrição do campo posicionado no Dicionário de Dados (SX3) no idioma corrente.
---------------	---

**Exemplo:**

```
User Function <nome-da-função>( )  
  
Local cTitulo  
Local cDescri  
Local cCombo  
  
dbSelectArea("SX3")  
dbSetOrder(2)  
  
If dbSeek( cCampo )  
cTitulo := X3Titulo()  
cDescri := X3Descri()  
cCombo := X3Cbox()  
EndIf  
  
Return
```

**X3PICTURE()**

A função X3PICTURE() retorna a máscara de um campo contido no Dicionário de Dados (SX3).

**Sintaxe: X3PICTURE(cCampo)**

**Parâmetros:**

<b>cCampo</b>	Nome do campo contido no Dicionário de Dados (SX3).
---------------	---

**Retorno:**

<b>String</b>	<i>Picture</i> do campo informado.
---------------	------------------------------------

**Exemplo:**

```
User Function <nome-da-função>( cCampo )  
  
Local cPicture  
  
cPicture := X3Picture( cCampo )  
Return cPicture
```

## X3TITULO()

---

A função X3TITULO() retorna o título de um campo posicionado no Dicionário de Dados (SX3) no idioma corrente.

### Sintaxe: X3TITULO()

#### Parâmetros:

Nenhum	( )
--------	-----

#### Retorno:

String	Título do campo posicionado no dicionário de dados (SX3) no idioma corrente.
--------	--

#### Exemplo:

```
User Function <nome-da-função>( )  
  
Local cTitulo  
  
dbSelectArea("SX3")  
dbSetOrder(2)  
  
If dbSeek( "A1_COD" )  
cTitulo := X3Titulo()  
Endif  
  
Return
```

## X3USO()

---

A função X3USO() verifica se o campo atualmente posicionado no Dicionário de Dados (SX3) está disponível para uso.

### Sintaxe: X3USO( cUsado, [Modulo] )

#### Parâmetros:

cUsado	Conteúdo do campo X3_USADO a ser avaliado.
Modulo	Número do módulo. Caso não seja informado será assumido como padrão o número do módulo corrente.

#### Retorno:

Lógico	Indica se o campo está configurado como usado no Dicionário de Dados (SX3).
--------	---

**Exemplo:**

```
User Function <nome-da-função>()  
  
Local lUsado := .F.  
  
DbSelectArea("SX3")  
DbSetOrder(2)  
DbSeek("A1_COD")  
  
If X3Uso( SX3->X3_USADO )  
lUsado := .T.  
EndIf  
  
Return lUsado
```

**X5DESCRI()**

---

A função X5DESCRI() retorna a descrição de um item de uma tabela posicionado no Arquivo de Tabelas (SX5) no idioma corrente.

**Sintaxe: X5DESCRI()****Parâmetros:**

<b>Nenhum</b>	()
---------------	----

**Retorno:**

<b>String</b>	Descrição do item do Arquivo de Tabelas (SX5) no idioma corrente.
---------------	---

**Exemplo:**

```
User Function <nome-da-função>( cFilial, cTabela, cChave )  
Local cDescr  
  
dbSelectArea("SX5")  
dbSetOrder(1)  
  
If dbSeek( cFilial+cTabela+cChave )  
cDescr := X5Descri()  
EndIf  
  
Return
```

## X6CONTEUD()

---

A função X6CONTEUD() retorna o conteúdo de um parâmetro posicionado no Dicionário de Dados (SX6) para o idioma corrente.

### Sintaxe: X6CONTEUD()

#### Parâmetros:

Nenhum	()
--------	----

#### Retorno:

Indefinido	Conteúdo do parâmetro posicionado no Dicionário de Dados (SX6) para o idioma corrente.
------------	--



*Importante*

Utilizar preferencialmente as funções de manipulação de parâmetros GETMV() e suas variantes.

---

#### Exemplo:

```
User Function <nome-da-função>( cFilial, cParam )
Local cDescr
Local cConteud

dbSelectArea("SX6")
dbSetOrder(1)

If dbSeek( cFilial+cParm )
cDescr := X6Descric()
cDescr += X6Desc1()
cDescr += X6Desc2()
cConteud := X6Conteud()
EndIf

Return
```

## X6DESCRIC()

---

A função X6DESCRIC() retorna o conteúdo da descrição de um parâmetro de acordo com o registro posicionado no Dicionário de Dados (SX6) no idioma corrente.

### Sintaxe: X6DESCRIC()

#### Parâmetros:

Nenhum	()
--------	----

#### Retorno:

String	Descrição do parâmetro posicionado no Dicionário de Dados (SX6) no idioma corrente.
--------	---

Para avaliar os conteúdos dos primeiro e segundo complementos da descrição do parâmetro utilize as funções:



Dica

- ☒ X6DESC01() ☒ retorna o primeiro complemento da descrição.
- ☒ X6DESC02() ☒ retorna o segundo complemento da descrição.

As três funções possuem a mesma sintaxe e forma de utilização.

---

### Exemplo:

```
User Function <nome-da-função>( cFilial, cParam )  
  
Local cDescr  
Local cConteud  
  
dbSelectArea("SX6")  
dbSetOrder(1)  
  
If dbSeek( cFilial+cParm )  
  cDescr := X6Descric()  
  cDescr += X6Desc1()  
  cDescr += X6Desc2()  
  cConteud := X6Conteud()  
EndIf  
Return
```

## XADESCRIC()

---

A função XADESCRIC() retorna o conteúdo da descrição das pastas de acordo com o registro posicionado no Dicionário de Dados (SXA) no idioma corrente.

### Sintaxe: XADESCRIC()

#### Parâmetros:

Nenhum	()
--------	----

#### Retorno:

String	Descrição do <i>folder</i> posicionado no Dicionário de Dados (SXA) no idioma corrente.
--------	---

#### Exemplo:

```
User Function <nome-da-função>( cFolder, cNumero )
Local cDescr
dbSelectArea("SXA")
dbSetOrder(1)
If dbSeek( cFolder+cNúmero ) // alias do folder + numero do folder
cDescr := XADescric()
EndIf
Return
```

## XBDESCRI()

---

A função XBDESCRI() retorna o conteúdo da descrição de uma consulta de acordo com o registro posicionado no Dicionário de Dados (SXB) no idioma corrente.

### Sintaxe: XBDESCRI()

#### Parâmetros:

Nenhum	()
--------	----

#### Retorno:

String	Descrição da consulta posicionada no Dicionário de Dados (SXB) no idioma corrente.
--------	--

**Exemplo:**

```
User Function <nome-da-função>( cAlias )
Local cDescr
dbSelectArea("SXB")
dbSetOrder(1)
If dbSeek( cAlias + "1" )
cDescr := XBDescri()
EndIf
Return
```

**XFILIAL()**

---

A função XFILIAL() retorna a filial utilizada por determinado arquivo.

Esta função é utilizada para permitir que pesquisas e consultas em arquivos trabalhem somente com os dados da filial corrente, dependendo, neste caso do compartilhamento do arquivo (definição que é feita através do módulo Configurador – Dicionário de Dados (SX2)).

É importante verificar que esta função não tem por objetivo retornar apenas a filial corrente, mas retorná-la caso o arquivo seja exclusivo. Se o arquivo estiver compartilhado, a função xFilial retornará dois espaços em branco.

**Sintaxe: XFILIAL(cAlias)**

**Parâmetros:**

<b>cAlias</b>	Alias do arquivo desejado. Se não for especificado, o arquivo tratado será o da área corrente.
---------------	--

**Retorno:**

<b>Caracter</b>	<i>String</i> contendo a filial do arquivo corrente.
-----------------	--

## Componentes da interface visual

---

### **MSDIALOG()**

---

Define o componente MSDIALOG(), o qual é utilizado como base para os demais componentes da interface visual, pois um componente MSDIALOG() é uma janela da aplicação.

**Sintaxe:**

```
DEFINE MSDIALOG oObjetoDLG TITLE cTitulo FROM nLinIni,nColIni TO  
nLiFim,nColFim OF oObjetoRef UNIDADE
```

**Parâmetros**

<b>oObjetoDLG</b>	Posição do objeto Say em função da janela em que ele será definido.
<b>cTitulo</b>	Título da janela de diálogo.
<b>nLinIni, nColIni</b>	Posição inicial em linha / coluna da janela.
<b>nLiFim, nColFim</b>	Posição final em linha / coluna da janela.
<b>oObjetoRef</b>	Objeto dialog no qual a janela será definida.
<b>UNIDADE</b>	Unidade de medida das dimensões: PIXEL

**Exemplo:**

```
DEFINE MSDIALOG oDlg TITLE cTitulo FROM 000,000 TO 080,300 PIXEL  
ACTIVATE MSDIALOG oDlg CENTERED
```

### **MSGET()**

---

Define o componente visual MSGET, o qual é utilizado para captura de informações digitáveis na tela da interface.

**Sintaxe:**

```
@ nLinha, nColuna MSGET VARIABEL SIZE nLargura,nAltura UNIDADE OF  
oObjetoRef F3 cF3 VALID VALID WHEN WHEN PICTURE cPicture
```

### Parâmetros

nLinha, nColuna	Posição do objeto MsGet em função da janela em que ele será definido.
VARIAVEL	Variável da aplicação que será vinculada ao objeto MsGet, que definirá suas características e na qual será armazenado o que for informado no campo.
nLargura,nAltura	Dimensões do objeto MsGet para exibição do texto.
UNIDADE	Unidade de medida das dimensões: PIXEL
oObjetoRef	Objeto dialog no qual o componente será definido.
cF3	String que define a consulta padrão que será vinculada ao campo.
VALID	Função de validação para o campo.
WHEN	Condição para manipulação do campo, a qual pode ser diretamente .T. ou .F., ou uma variável ou uma chamada de função.
cPicture	String contendo a definição da Picture de digitação do campo.

### Exemplo:

```
@ 010,050 MSGET cCGC      SIZE 55, 11 OF oDlg PIXEL PICTURE "@R  
99.999.999/9999-99";  
VALID !Vazio()
```

### SAY()

Define o componente visual SAY, o qual é utilizado para exibição de textos em uma tela de interface.

### Sintaxe:

```
@ nLinha, nColuna SAY cTexto  SIZE nLargura,nAltura UNIDADE OF  
oObjetoRef
```

### Parâmetros

nLinha, nColuna	Posição do objeto Say em função da janela em que ele será definido.
cTexto	Texto que será exibido pelo objeto Say.
nLargura,nAltura	Dimensões do objeto Say para exibição do texto.
UNIDADE	Unidade de medida das dimensões: PIXEL
oObjetoRef	Objeto dialog no qual o componente será definido.

**Exemplo:**

```
@ 010,010 SAY    cTexto SIZE 55, 07 OF oDlg PIXEL
```

**BUTTON()**

Define o componente visual Button, o qual permite a inclusão de botões de operação na tela da interface, os quais serão visualizados somente com um texto simples para sua identificação.

**Sintaxe: BUTTON()**

```
@ nLinha,nColuna BUTTON cTexto SIZE  nLargura,nAltura UNIDADE OF  
oObjetoRef  
ACTION AÇÃO
```

**Parâmetros**

nLinha,nColuna	Posição do objeto Button em função da janela em que ele será definido.
cTexto	String contendo o texto que será exibido no botão.
nLargura,nAltura	Dimensões do objeto Button para exibição do texto.
UNIDADE	Unidade de medida das dimensões: PIXEL.
oObjetoRef	Objeto dialog no qual o componente será definido.
AÇÃO	Função ou lista de expressões que define o comportamento do botão quando ele for utilizado.

**Exemplo:**

```
010, 120 BUTTON "Confirmar"  SIZE 080, 047 PIXEL OF oDlg;  
ACTION (nOpc := 1,oDlg:End())
```

## **SBUTTON()**

Define o componente visual SButton, o qual permite a inclusão de botões de operação na tela da interface, os quais serão visualizados dependendo da interface do sistema ERP utilizada somente com um texto simples para sua identificação, ou com uma imagem (BitMap) pré-definido.

### **Sintaxe: SBUTTON()**

```
DEFINE SBUTTON FROM nLinha, nColuna TYPE N ACTION AÇÃO STATUS OF  
oObjetoRet
```

### **Parâmetros**

nLinha, nColuna	Posição do objeto sButton em função da janela em que ele será definido.
TYPE N	Número que indica o tipo do botão (imagem) pré-definida que será utilizada.
AÇÃO	Função ou lista de expressões que define o comportamento do botão quando ele for utilizado.
STATUS	Propriedade de uso do botão: ENABLE ou DISABLE
oObjetoRet	Objeto dialog no qual o componente será definido.

### **Exemplo:**

```
DEFINE SBUTTON FROM 020, 120 TYPE 2 ACTION (nOpcA := 2,oDlg:End());  
ENABLE OF oDlg
```

### **Visual dos diferentes tipos de botões disponíveis**



## **CHECKBOX()**

---

Define o componente visual CheckBox, o qual permite a utilização da uma marca para habilitar ou não uma opção escolhida, sendo esta marca acompanhada de um texto explicativo. Difere do RadioMenu pois cada elemento do check é único, mas o Radio permite a utilização de uma lista junto com um controle de seleção.

### **Sintaxe:**

```
@ nLinha,nColuna CHECKBOX oCheckBox VAR VARIABEL PROMPT cTexto WHEN  
WHEN UNIDADE OF oObjetoRef SIZE nLargura,nAltura MESSAGE cMensagem
```

### **Parâmetros:**

<b>nLinha,nColuna</b>	Posição do objeto ComboBox em função da janela em que ele será definido.
<b>oCheckBox</b>	Objeto do tipo CheckBox que será criado.
<b>VARIABEL</b>	Variável do tipo lógico com o status do objeto (.T. – marcado, .F. – desmarcado).
<b>cTexto</b>	Texto que será exibido ao lado do get de marcação.
<b>WHEN</b>	Condição para manipulação do objeto, a qual pode ser diretamente .T. ou .F., ou uma variável ou uma chamada de função.
<b>UNIDADE</b>	Unidade de medida das dimensões: PIXEL.
<b>oObjetoRef</b>	Objeto <i>dialog</i> no qual o componente será definido.
<b>nLargura,nAltura</b>	Dimensões do objeto CheckBox.
<b>cMensagem</b>	Texto que será exibido ao clicar no componente.

### **Exemplo:**

```
@ 110,10 CHECKBOX oChk VAR lChk PROMPT "Marca/Desmarca" SIZE 60,007  
PIXEL OF oDlg ;  
ON CLICK(aEval(aVetor,{|x| x[1]:=lChk}),oLbx:Refresh())
```

## **COMBOBOX()**

---

Define o componente visual ComboBox, o qual permite seleção de um item dentro de uma lista de opções de textos simples no formato de um vetor.

**Sintaxe:**

```
@ nLinha,nColuna COMBOBOX VARIABEL ITEMS AITENS SIZE  
nLargura,nAltura UNIDADE OF oObjetoRef
```

**Parâmetros:**

<b>nLinha,nColuna</b>	Posição do objeto ComboBox em função da janela em que ele será definido.
<b>VARIABEL</b>	Variável do tipo caracter que irá receber a descrição do item selecionado no ComboBox.
<b>AITENS</b>	Vetor simples contendo as <i>strings</i> que serão exibidas como opções do ComboBox.
<b>nLargura,nAltura</b>	Dimensões do objeto ComboBox.
<b>UNIDADE</b>	Unidade de medida das dimensões: PIXEL.
<b>oObjetoRef</b>	Objeto dialog no qual o componente será definido.

**Exemplo:**

```
@ 40, 10 COMBOBOX oCombo VAR cCombo ITEMS aCombo SIZE 180,10 PIXEL  
OF oFlId:aDialogs[2]
```

## **FOLDER()**

---

Define o componente *visual Folder*, o qual permite a inclusão de diversos *Dialogs* dentro de uma mesma interface visual. Um *Folder* pode ser entendido como um array de *Dialogs*, aonde cada painel recebe seus componentes e tem seus atributos definidos independentemente dos demais.

**Sintaxe:**

```
@ nLinha,nColuna FOLDER oFolder OF oObjetoRef PROMPT  
&cTexto1,...,&cTextoX UNIDADE SIZE nLargura,nAltura
```

## Parâmetros

<b>nLinha,nColuna</b>	Posição do objeto <i>Folder</i> em função da janela em que ele será definido.
<b>oFolder</b>	Objeto <i>Folder</i> que será criado.
<b>oObjetoRef</b>	Objeto <i>dialog</i> no qual o componente será definido.
<b>&amp;cTexto1,...,&amp;cTextoX</b>	Strings de títulos de cada uma das abas do <i>Folder</i> , sempre precedidas por &. Exemplo: "&Pasta1","&PastaX".
<b>UNIDADE</b>	Unidade de medida das dimensões: PIXEL
<b>nLargura,nAltura</b>	Dimensões do objeto <i>Folder</i> .

### Exemplo:

```
@ 50,06 FOLDER oFd OF oDlg PROMPT "&Buscas", "&Consultas", "Check-&Up / Botões" PIXEL SIZE 222,078
```

## RADIO()

Define o componente visual Radio, também conhecido como RadioMenu, o qual é seleção de uma opção ou de múltiplas opções através de uma marca para os itens exibidos de uma lista. Difere do componente *CheckBox*, pois cada elemento de *check* é sempre único, e o Rádio pode conter um ou mais elementos.

### Sintaxe:

```
@ nLinha,nColuna RADIO oRadio VAR nRadio 3D SIZE nLargura,nAltura
<ITEMS PROMPT> cItem1,cItem2,...,cItemX OF oObjetoRef UNIDADE ON
CHANGE CHANGE ON CLICK CLICK
```

## Parâmetros

<b>nLinha,nColuna</b>	Posição do objeto Rádio em função da janela em que ele será definido.
<b>oRadio</b>	Objeto do tipo Rádio que será criado.
<b>nRadio</b>	Item do objeto Rádio que está selecionado.
<b>3D</b>	Item opcional que define se o RadioButton terá aspecto simples ou 3D.
<b>nLargura,nAltura</b>	Dimensões do objeto Rádio.
<b>&lt;ITEMS PROMPT&gt;</b>	Utilizar um dos dois identificadores para definir quais os textos que serão vinculados a cada RadioButton.
<b>cItem1,cItem2,...,cItemX</b>	Texto que será vinculado a cada RadioButton.
<b>oObjetoRef</b>	Objeto dialog no qual o componente será definido.
<b>UNIDADE</b>	Unidade de medida das dimensões: PIXEL
<b>CHANGE</b>	Função ou lista de expressões que será executada na mudança de um item de um RadioButton para outro.
<b>CLICK</b>	Função ou lista de expressões que será executada na seleção de um item RadioButton.

### Exemplo:

```

aAdd( aRadio, "Disco" )
aAdd( aRadio, "Impressora" )
aAdd( aRadio, "Scanner" )

@ 30, 10 RADIO oRadio VAR nRadio ITEMS aRadio[1],aRadio[2],aRadio[3]
SIZE 65,8 ;
PIXEL OF ;
    oFlx:aDialogs[3] ;
    ON CHANGE ;
        (lif(nRadio==1,MsgInfo("Opcão 1",cAtencao),;
            lif(nRadio==2,MsgInfo("Opcão 2",cAtencao),MsgInfo("Opcão
3",cAtencao))))
```

## Interfaces de cadastro

---

### AXCADASTRO()

---

<b>Sintaxe</b>	<b>AxCadastro(cAlias, cTitulo, cVldExc, cVldAlt)</b>
<b>Descrição</b>	O AxCadastro() é uma funcionalidade de cadastro simples, com poucas opções de customização.

### MBROWSE()

---

<b>Sintaxe</b>	<b>MBrowse(nLin1, nCol1, nLin2, nCol2, cAlias)</b>
<b>Descrição</b>	A Mbrowse() é uma funcionalidade de cadastro que permite a utilização de recursos mais aprimorados na visualização e manipulação das informações do sistema.

### AXPESQUI()

---

Função de pesquisa padrão em registros exibidos pelos *browses* do sistema, a qual posiciona o *browse* no registro pesquisado. Exibe uma tela que permite a seleção do índice a ser utilizado na pesquisa e a digitação das informações que compõe a chave de busca.

#### Sintaxe: AXPESQUI()

#### Parâmetros

<b>Nenhum</b>	<b>()</b>
---------------	-----------

### AXVISUAL()

---

Função de visualização padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

#### Sintaxe: AXVISUAL(cAlias, nReg, nOpc, aAcho, nColMens, cMensagem, cFunc,; aButtons, lMaximized )

## Parâmetros

<b>cAlias</b>	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada
<b>nReg</b>	<i>Record number</i> (recno) do registro posicionado no alias ativo.
<b>nOpc</b>	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização).
<b>aAcho</b>	Vetor com nome dos campos que serão exibidos. Os campos de usuário sempre serão exibidos se não existir no parâmetro um elemento com a expressão "NOUSER".
<b>nColMens</b>	Parâmetro não utilizado.
<b>cMensagem</b>	Parâmetro não utilizado.
<b>cFunc</b>	Função que deverá ser utilizada para carregar as variáveis que serão utilizadas pela Enchoice. Neste caso o parâmetro lVirtual é definido internamente pela AxFunction() executada como .T.
<b>aButtons</b>	Botões adicionais para a EnchoiceBar, no formato: aArray[n][1] -> Imagem do botão aArray[n][2] -> bloco de código contendo a ação do botão aArray[n][3] -> título do botão
<b>lMaximized</b>	Indica se a janela deverá ser ou não maximizada

## AXINCLUI()

Função de inclusão padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

**Sintaxe:** AxInclui(cAlias, nReg, nOpc, aAcho, cFunc, aCpos, cTudoOk, IF3,;  
cTransact, aButtons, aParam, aAuto, lVirtual, lMaximized)

## Parâmetros

<b>cAlias</b>	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada.
<b>nReg</b>	<i>Record number</i> (recno) do registro posicionado no alias ativo.
<b>nOpc</b>	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização).
<b>aAcho</b>	Vetor com nome dos campos que serão exibidos. Os campos de usuário sempre serão exibidos se não existir no parâmetro um elemento com a expressão "NOUSER".
<b>cFunc</b>	Função que deverá ser utilizada para carregar as variáveis que serão utilizadas pela Enchoice. Neste caso, o parâmetro lVirtual é definido internamente pela AxFunction() executada como .T.
<b>aCpos</b>	Vetor com nome dos campos que poderão ser editados.
<b>cTudoOk</b>	Função de validação de confirmação da tela. Não deve ser passada como Bloco de Código, mas pode ser passada como uma lista de expressões,

	desde que a última ação efetue um retorno lógico: “(Func1(), Func2(), ..., FuncX(), .T. )”
<b>IF3</b>	Indica se a enchoice está sendo criada em uma consulta F3 para utilizar variáveis de memória.
<b>cTransact</b>	Função que será executada dentro da transação da AxFunction().
<b>aButtons</b>	Botões adicionais para a EnchoiceBar, no formato: aArray[n][1] -> Imagem do botão aArray[n][2] -> bloco de código contendo a ação do botão aArray[n][3] -> título do botão
<b>aParam</b>	Funções para execução em pontos pré-definidos da AxFunction(), conforme abaixo: aParam[1] := Bloco de código que será processado antes da exibição da interface. aParam[2] := Bloco de código para processamento na validação da confirmação. aParam[3] := Bloco de código que será executado dentro da transação da AxFunction(). aParam[4] := Bloco de código que será executado fora da transação da AxFunction().
<b>aAuto</b>	Array no formato utilizado pela funcionalidade MsExecAuto(). Caso seja informado este array, não será exibida a tela de interface, e será executada a função EnchAuto(). aAuto[n][1] := Nome do campo aAuto[n][2] := Conteúdo do campo aAuto[n][3] := Validação que será utilizada em substituição as validações do SX3
<b>IVirtual</b>	Indica se a Enchoice() chamada pela AxFunction() utilizará variáveis de memória ou os campos da tabela na edição.
<b>IMaximized</b>	Indica se a janela deverá ser ou não maximizada.

## **AXALTERA()**

---

Função de alteração padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

.

**Sintaxe:** AXALTERA(cAlias, nReg, nOpc, aAcho, aCpos, nColMens, cMensagem,; cTodoOk, cTransact, cFunc, aButtons, aParam, aAuto, IVirtual, IMaximized)

.

**Parâmetros**

□ Vide documentação de parâmetros da função AxInclui().

## AXDELETA()

Função de exclusão padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

**Sintaxe: AXDELETA(cAlias, nReg, nOpc, cTransact, aCpos, aButtons, aParam,; aAuto, lMaximized)**

### Parâmetros

<b>cAlias</b>	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada.
<b>nReg</b>	Record number (recno) do registro posicionado no alias ativo.
<b>nOpc</b>	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização).
<b>cTransact</b>	Função que será executada dentro da transação da AxFunction().
<b>aCpos</b>	Vetor com nome dos campos que poderão ser editados.
<b>aButtons</b>	Botões adicionais para a EnchoiceBar, no formato: aArray[n][1] -> Imagem do botão aArray[n][2] -> bloco de código contendo a ação do botão aArray[n][3] -> título do botão
<b>aParam</b>	Funções para execução em pontos pré-definidos da AxFunction(), conforme abaixo: aParam[1] := Bloco de código que será processado antes da exibição da interface. aParam[2] := Bloco de código para processamento na validação da confirmação. aParam[3] := Bloco de código que será executado dentro da transação da AxFunction(). aParam[4] := Bloco de código que será executado fora da transação da AxFunction().
<b>aAuto</b>	Array no formato utilizado pela funcionalidade MsExecAuto(). Caso seja informado este array, não será exibida a tela de interface, e será executada a função EnchAuto(). aAuto[n][1] := Nome do campo aAuto[n][2] := Conteúdo do campo aAuto[n][3] := Validação que será utilizada em substituição as validações do SX3
<b>lMaximized</b>	Indica se a janela deverá ser ou não maximizada.

## Interfaces visuais para aplicações

### **ALERT()**

---

Sintaxe: AVISO(cTexto)

#### **Parâmetros**

cTexto	Texto a ser exibido
	

### **AVISO()**

---

Sintaxe: AVISO(cTitulo, cTexto, aBotoes, nTamanho)

Retorno: numérico indicando o botão selecionado

#### **Parâmetros**

cTitulo	Título da janela.
cTexto	Texto do aviso.
aBotoes	Array simples (vetor) com os botões de opção.
nTamanho	Tamanho (1,2 ou 3).
	

## **FORMBACTH()**

---

**Sintaxe:** FORMBATC $\text{H}$ (cTítulo, aTexto, aBotoes, bValid, nAltura, nLargura )

### **Parâmetros**

<b>cTítulo</b>	Título da janela.
<b>aTexto</b>	Array simples (vetor) contendo cada uma das linhas de texto que serão exibidas no corpo da tela.
<b>aBotoes</b>	Array com os botões do tipo SBUTTON(), com a seguinte estrutura:  <code>{nTipo, lEnable, {   Ação() }}</code>
<b>bValid</b>	(opcional) Bloco de validação da janela.
<b>nAltura</b>	(opcional) Altura em pixels da janela.
<b>nLargura</b>	(opcional) Largura em pixels da janela.
	

## MSGFUNCTIONS()

Sintaxe: MSGALERT(cTexto, cTitulo)

Sintaxe: MSGINFO(cTexto, cTitulo)

Sintaxe: MSGSTOP(cTexto, cTitulo)

Sintaxe: MSGYESNO(cTexto, cTitulo)

Parâmetros

cTexto	Texto a ser exibido como mensagem.
cTitulo	Título da janela de mensagem.
MSGALERT	
MSGINFO	
MSGSTOP	
MSGYESNO	

## **Recursos das interfaces visuais**

### **GDFIELDGET()**

---

A função GDFIELDGET() retorna o conteúdo de um campo especificado em uma *grid* formada por um objeto do tipo MsNewGetDados() de acordo com a linha da *grid* desejada.

**Sintaxe:** **GDFIELDGET(cCampo, nLinha)**

**Parâmetros:**

<b>cCampo</b>	Nome do campo para retorno do conteúdo.
<b>nLinha</b>	Linha da <i>grid</i> que deverá ser avaliada.

**Retorno:**

<b>Indefinido</b>	Conteúdo do campo especificado de acordo com a linha da <i>grid</i> informada.
-------------------	--

### **GDFIELDPOS()**

---

A função GDFIELDPOS() retorna a posição de um campo especificado em uma *grid* formada por um objeto do tipo MsNewGetDados().

**Sintaxe:** **GDFIELDPOS(cCampo)**

**Parâmetros:**

<b>cCampo</b>	Nome do campo a ser avaliado na <i>grid</i> .
---------------	---

**Retorno:**

<b>Numérico</b>	Posição que o campo ocupada na <i>grid</i> . Caso o mesmo não exista será retornado 0.
-----------------	--

### **GDFIELDPUT()**

---

A função GDFIELDPUT() atualiza o conteúdo de uma *grid* formada por um objeto do tipo MsNewGetDados() de acordo com o campo e linha da *grid* especificados.

**Sintaxe:** **GDFIELDPUT(cCampo, xConteudo, nLinha)**

**Parâmetros:**

cCampo	Nome do campo a ser atualizado.
xConteúdo	Conteúdo que será atribuído a célula da <i>grid</i> .
nLinha	Linha da grid que será atualizada.

**Retorno:**

Nenhum	()
--------	----

**GETMARK()**

---

A função GETMARK() é utilizada em conjunto com a função MarkBrow(), para retornar o conjunto de caracteres que serão utilizados para identificar os registros marcados pelo *browse*.

**Sintaxe:** GETMARK( [IUpper] )

**Parâmetros:**

IUpper	Se verdadeiro (.T.) retorna somente caracteres em maiúsculos.
--------	---

**Retorno:**

String	Conjunto de caracteres que definem a marca que deverá ser utilizada pela MarkBrowse durante o processamento corrente.
--------	---



*Importante*

O retorno da função GETMARK() depende do conteúdo atual do parâmetro MV\_MARCA.

---



*Importante*

É altamente recomendável limpar o conteúdo do campo “marcado” pela MarkBrowse() ao término do processamento, para se evitar problemas com a reutilização da marca após a exaustão das possibilidades de combinação de dois caracteres, o qual é o tamanho padrão dos campos utilizados para marcação de registros pela MarkBrowse(), que neste caso somam **1891** possibilidades de “00” a “zz”.

---

**Exemplo:**

```
Function <nome-da-função>()

Local aCampos := {{'CB_OK' ,,""},;
{'CB_USERLIB' ,,'Usuário'},;
{'CB_TABHORA' ,,'Hora'},;
{'CB_DTTAB' ,,'Data'}}

Private cMarca := GetMark()
Private cCadastro := 'Cadastro de Contrato'
Private aRotina := { { 'Pesquisar' , 'AxPesqui' , 0, 1 } }

MarkBrow( 'SCB', 'CB_OK','!CB_USERLIB',aCampos,,,
cMarca,'MarkAll()',,,,'Mark()' )

Return
```

---

**MARKBREFRESH()**

---

A função MARKBREFRESH() atualiza a exibição da marca no MarkBrowse(), sendo utilizada quando algum processamento paralelo atualiza o conteúdo do campo definido como controle de marca para os registros em exibição pelo *browse*.

Este tipo de processamento é comum, e normalmente está associada a clique de “inverter” seleção, ou a opções de “marcar” e “desmarcar” todas.

---



*Importante*

A MarkBrowse() atualiza automaticamente a exibição da marca de registros quando utilizado o browse.

---

**Sintaxe: MARKBREFRESH()****Parâmetros:**

<b>Nenhum</b>	()
---------------	----

**Retorno:**

<b>Nenhum</b>	()
---------------	----

## **READVAR()**

---

A função READVAR() retorna o nome da variável ou campo associado ao objeto do tipo GET() atualmente selecionado ou em edição.

### **Sintaxe: READVAR()**

#### **Parâmetros:**

<b>Nenhum</b>	<b>()</b>
---------------	-----------

#### **Retorno:**

<b>String</b>	Nome da variável ou campo associado ao objeto do tipo GET.
---------------	--

**ADVPL1**

## **Conteúdo**

OBJETIVOS DO CURSO	487
MÓDULO 01: INTRODUÇÃO À PROGRAMAÇÃO	488
1.2. Desenvolvendo algoritmos	491
Regras para a construção do Algoritmo	492
1.2.2.	495
Teste de mesa	495
2. ESTRUTURAS DE PROGRAMAÇÃO	497
2.2. Estruturas de decisão e repetição	502
2.2.2.	507
Estruturas de repetição	507
MÓDULO 02: A LINGUAGEM ADVPL	511
3. ESTRUTURA DE UM PROGRAMA ADVPL	514
3.1. Áreas de um Programa ADVPL	518
4. DECLARAÇÃO E ATRIBUIÇÃO DE VARIÁVEIS	524
4.2. Declaração de variáveis	526
4.3. Escopo de variáveis	528
4.4. Entendendo a influência do escopo das variáveis	537
4.5. Operações com Variáveis	539
4.5.2.	541
Operadores da linguagem ADVPL	541
Operadores de Atribuição	544
Operadores Especiais	547
Ordem de Precedência dos Operadores	549
4.5.3.	551
Operação de Macro Substituição	551
4.5.4.	553
Funções de manipulação de variáveis	553
Verificação de tipos de variáveis	561
5. ESTRUTURAS BÁSICAS DE PROGRAMAÇÃO	562
5.1. Estruturas de repetição	562
5.1.1.	568
Influenciando o fluxo de repetição	568
5.2. Estruturas de decisão	570

6. ARRAYS E BLOCOS DE CÓDIGO	579
6.1.1.	583
Inicializando arrays	583
6.1.2.	586
Funções de manipulação de arrays	586
6.1.3.	589
Cópia de arrays	589
6.2. Listas de Expressões e Blocos de Código	594
6.2.2.	596
Lista de expressões	596
Convertendo para uma lista de expressões	597
6.2.3.	600
Blocos de Código	600
6.2.4.	603
Funções para manipulação de blocos de código	603
7. FUNÇÕES	605
7.1. Tipos e escopos de funções	607
Static Function()	610
7.2. Passagem de parâmetros entre funções	612
Passagem de parâmetros por referência	617
8. DIRETIVAS DE COMPILAÇÃO	622
Diretiva: #DEFINE	627
Diretivas: #IFDEF, IFNDEF, #ELSE e #ENDIF	628
Diretiva: #COMMAND	630
9. ADVPL E O ERP MICROSIGA PROTHEUS	635
9.1. O Ambiente Protheus	636
9.2. Organização e configuração inicial do ambiente Protheus	641
Propriedades dos atalhos	646
9.3. O Configurador do Protheus	651
9.3.1.	651
Funcionalidades Abordadas	651
9.3.2.	651
Estruturas básicas da aplicação ERP Protheus	651
Ambientes e tabelas	653
Índices	658

9.3.3.	659
Acessando o módulo Configurador	659
9.4. Funcionalidades do Configurador	661
9.4.1.	663
Dicionário de Dados da aplicação ERP	663
9.4.2.	664
Adição de tabelas ao Dicionário de Dados	664
9.4.3.	668
Adição de campos às tabelas do Dicionário de Dados	668
9.4.4.	676
Adição de índices para as tabelas do Dicionário de Dados	676
Orientações para o cadastramento de um novo índice	681
9.4.6.	685
Criação de Tabelas Genéricas	685
9.4.7.	687
Criação de Parâmetros	687
10. TOTVS DEVELOPMENT STUDIO	689
DESENVOLVIMENTO DE PEQUENAS CUSTOMIZAÇÕES	693
11.1. Diferenças e compatibilizações entre bases de dados	695
11.2. Funções de acesso e manipulação de dados	697
11.3. Diferenciação entre variáveis e os nomes de campos	706
11.4. Controle de numeração sequencial	708
Funções de controle de semáforos e numeração sequencial	709
12. CUSTOMIZAÇÕES PARA A APLICAÇÃO ERP	710
12.1. Customização de campos – Dicionário de Dados	711
12.1.1.	712
Validações de campos e perguntas	712
12.1.2.	715
Pictures de formação disponíveis	715
12.2. Customização de gatilhos – Configurador	719
12.3. Customização de parâmetros – Configurador	720
12.3.1.	721
Funções para a manipulação de parâmetros	721
12.3.2.	723
Cuidados na utilização de um parâmetro	723

12.4. Pontos de Entrada – Conceitos, Premissas e Regras	724
13. INTERFACES VISUAIS	726
13.1. Sintaxe e componentes das interfaces visuais	726
13.2. Interfaces padrões para atualizações de dados	730
13.2.1. MBrowse()	732
13.2.2. AxFunctions()	738
13.2.3. ADVPL	738
14. UTILIZAÇÃO DE IDENTAÇÃO	742
15. CAPITULAÇÃO DE PALAVRAS-CHAVE	744
15.1. Palavras em maiúsculo	745
17. PALAVRAS RESERVADAS	747
GUIA DE REFERÊNCIA RÁPIDA: Funções e Comandos	749
Verificação de tipos de variáveis	755
Manipulação de blocos de código	770
Manipulação de strings	776
Manipulação de variáveis numéricas	783
Manipulação de arquivos	788
Controle de numeração sequencial	808
Validação	809
GETMV()	812
Componentes da interface visual	816
Interfaces de cadastro	822
Funções visuais para aplicações	829
Funções ADVPL para aplicações	834
REFERÊNCIAS BIBLIOGRÁFICAS	836

## **OBJETIVOS DO CURSO**

Objetivos específicos do curso:

Ao final do curso o treinando deverá ter desenvolvido os seguintes conceitos, habilidades e atitudes:

a) Conceitos a serem aprendidos

Fundamentos e técnicas de programação;  
Princípios básicos da linguagem ADVPL;  
Comandos e funções específicas da Microsiga.

b) Habilidades e técnicas a serem aprendidas

Resolução de algoritmos através de sintaxes orientadas a linguagem ADVPL;  
Análise de fontes de baixa complexidade da aplicação ERP Protheus;  
Desenvolvimento de pequenas customizações para o ERP Protheus.

c) Atitudes a serem desenvolvidas

Adquirir conhecimentos através da análise das funcionalidades disponíveis no ERP Protheus;  
Embasar a realização de outros cursos relativos à linguagem ADVPL.

## **MÓDULO 01: INTRODUÇÃO À PROGRAMAÇÃO**

### **1. LÓGICA DE PROGRAMAÇÃO E ALGORITMOS**

No aprendizado de qualquer linguagem de programação é essencial desenvolver os conceitos relacionados à lógica e à técnica da escrita de um programa.

Com foco nesta necessidade, este tópico descreverá resumidamente os conceitos envolvidos no processo de desenvolvimento de um programa, através dos conceitos relacionados a:

- Lógica de programação
- Algoritmos
- Diagramas de blocos

#### **1.1. Lógica de Programação**

##### **Lógica**

A lógica de programação é necessária para pessoas que desejam trabalhar com desenvolvimento de sistemas e programas. Ela permite definir a sequência lógica para o desenvolvimento. Então o que é lógica?

Lógica de programação é a técnica de encadear pensamentos para atingir determinado objetivo.

##### **Sequência Lógica**

Estes pensamentos podem ser descritos como uma sequência de instruções, que devem ser seguidas para que seja cumprida determinada tarefa.

Sequência Lógica são passos executados até atingir um objetivo, ou solução de um problema.

## Instruções

---

Na linguagem comum entende-se por instruções como “um conjunto de regras, ou normas definidas para a realização, ou emprego de algo”. Em informática, porém, instrução é a informação que indica a um computador uma ação elementar a executar. Convém ressaltar que uma ordem isolada não permite realizar o processo completo, para isso é necessário um conjunto de instruções colocadas em ordem seqüencial lógica.

Por exemplo, se quisermos fazer uma omelete de batatas, precisaremos colocar em prática uma série de instruções: descascar as batatas, bater os ovos, fritar as batatas, etc. É evidente que essas instruções devem ser executadas em uma ordem adequada – não se podem descascar as batatas depois de fritá-las.

Dessa maneira, uma instrução tomada isoladamente não tem muito sentido para obtermos o resultado. Precisamos colocar em prática o conjunto de todas as instruções, e na ordem correta.

Instruções é um conjunto de regras, ou normas definidas para a realização ou emprego de algo. Em informática, é o que indica a um computador uma ação elementar a executar.

## Algoritmo

---

Um algoritmo é formalmente uma sequência finita de passos que levam à execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma sequência de instruções que dão cabo de uma meta específica. Estas tarefas não podem ser redundantes nem subjetivas na sua definição, devem ser claras e precisas.

Como exemplos de algoritmos é possível citar os algoritmos das operações básicas (adição, multiplicação, divisão e subtração) de números reais decimais. Outros exemplos seriam os

manuais de aparelhos eletrônicos que explicam passo-a-passo como, por exemplo, gravar um evento.

Até mesmo as coisas mais simples podem ser descritas por sequências lógicas, tais como:

“Chupar uma bala”

1. Pegar a bala;
2. Retirar o papel;
3. Chupar a bala;
4. Jogar o papel no lixo.

“Somar dois números quaisquer”

1. Escreva o primeiro número no retângulo A;
2. Escreva o segundo número no retângulo B;
3. Some o número do retângulo A com número do retângulo B e coloque o resultado no retângulo C.

## **1.2. Desenvolvendo algoritmos**

### Pseudocódigo

Os algoritmos são descritos em uma linguagem chamada pseudocódigo. Este nome é uma alusão à posterior implementação em uma linguagem de programação, ou seja, quando for utilizada a linguagem de programação propriamente dita como, por exemplo, ADVPL.

Por isso os algoritmos são independentes das linguagens de programação, sendo que ao contrário de uma linguagem de programação, não existe um formalismo rígido de como deve ser escrito o algoritmo.

O algoritmo deve ser fácil de interpretar e fácil de codificar. Ou seja, ele deve ser o intermediário entre a linguagem falada e a linguagem de programação.

## **Regras para a construção do Algoritmo**

Para escrever um algoritmo precisamos descrever a sequência de instruções, de maneira simples e objetiva. Para isso utilizaremos algumas técnicas:

1. Usar somente um verbo por frase;
2. Imaginar que você está desenvolvendo um algoritmo para pessoas que não trabalham com Informática;
3. Usar frases curtas e simples;
4. Ser objetivo;
5. Procurar usar palavras que não tenham duplo sentido.

### Fases

Para implementar um algoritmo de simples interpretação e codificação é necessário inicialmente dividir o problema apresentado em três fases fundamentais:

**ENTRADA:** são os dados de entrada do algoritmo;

**PROCESSAMENTO:** são os procedimentos utilizados para chegar ao resultado final;

**SAÍDA:** são os dados já processados.

Neste tópico serão demonstrados alguns algoritmos do cotidiano que foram implementados utilizando os princípios descritos nos tópicos anteriores.

Mascar um chiclete

Utilizar um telefone público – cartão

Fritar um ovo

Trocar lâmpadas

Descascar batatas

Jogar o jogo da forca

Calcular a média de notas

Jogar o jogo da velha – contra o algoritmo

## Mascar um chiclete

---

1. Pegar o chiclete.
2. Retirar o papel.
3. Mastigar.
4. Jogar o papel no lixo.

## Utilizar um telefone público - cartão

---

1. Retirar o telefone do gancho.
2. Esperar o sinal.
3. Colocar o cartão.
4. Discar o número.
5. Falar no telefone.
6. Colocar o telefone no gancho.

## Fritar um ovo

---

1. Pegar frigideira, ovo, óleo e sal.
2. Colocar óleo na frigideira.
3. Ascender o fogo.
4. Colocar a frigideira no fogo.
5. Esperar o óleo esquentar.
6. Quebrar o ovo na frigideira.
7. Jogar a casca no lixo.
8. Retirar a frigideira do fogo quando o ovo estiver no ponto.
9. Desligar o fogo.
10. Colocar sal a gosto.

## Trocar lâmpadas

---

1. Se a lâmpada estiver fora do alcance, pegar uma escada.
2. Pegar a lâmpada nova.
3. Se a lâmpada queimada estiver quente, pegar um pano.
4. Tirar a lâmpada queimada.
5. Colocar a lâmpada nova.

## **Descascar batatas**

---

1. Pegar faca, bacia e batatas.
2. Colocar a água na bacia.
3. Descascar todas as batatas.
  - 3.1. Colocar as batatas descascadas na bacia.

## **Jogar o jogo da forca**

---

1. Escolher a palavra.
2. Montar o diagrama do jogo.
3. Enquanto houver lacunas vazias e o corpo estiver incompleto:
  - 3.1. Se acertar a letra: escrever na lacuna correspondente.
  - 3.2. Se errar a letra: desenhar uma parte do corpo na forca.

## **Calcular a média de notas**

---

1. Enquanto houver notas a serem recebidas:
  - 1.1. Receber a nota.
2. Some todas as notas recebidas.
3. Divida o total obtido pela quantidade de notas recebidas.
4. Exiba a média das notas.

## **Jogar o jogo da velha – contra o algoritmo**

---

1. Enquanto existir um quadrado livre e ninguém ganhou ou perdeu o jogo:
  - 1.1. Espere a jogada do adversário, continue depois.
  - 1.2. Se o centro estiver livre: jogue no centro.
  - 1.3. Senão, se o adversário possuir dois quadrados em linha com um quadrado livre jogue neste quadrado.
  - 1.4. Senão, se há algum canto livre, jogue neste canto.

## 1.2.2.

### Teste de mesa

Todo algoritmo desenvolvido, deve ser testado. Este teste é chamado “TESTE DE MESA”, que significa seguir as instruções do algoritmo de maneira precisa para verificar se o procedimento utilizado está correto ou não.

Para avaliar a aplicação do teste de mesa, utilizaremos o algoritmo de calcular a média de notas:

- Algoritmo: Calcular a média de notas

1. Enquanto houver notas a serem recebidas:
  - a. Receber a nota.
2. Some todas as notas recebidas.
3. Divida o total obtido pela quantidade de notas recebidas.
4. Exiba a média das notas.

Teste de mesa:

1. Para cada nota informada, receber e registrar na tabela abaixo:

ID	Nota
----	------

2. Ao término das notas, a tabela deverá conter todas as notas informadas, como abaixo:

ID	Nota
1	8.0
2	7.0

4	8.0
5	7.0
6	7.0

3. Some todas as notas: 45  
 4. Divida a soma das notas, pelo total de notas informado:  $45/6 \quad 7.5$   
 5. Exiba a média obtida: Mensagem (Média: 7.5)



### Anotações

---



---



---



---

### Exercícios

#### Exercício 01

Vamos aprimorar os seguintes algoritmos descritos na apostila:

- 1) Usar telefone público – cartão.
- 2) Fritar um ovo.
- 3) Mascar um chiclete
- 4) Trocar lâmpadas.
- 5) Descascar batatas.
- 6) Usar telefone público – cartão
- 7) Jogar o “Jogo da Forca”

## **2. ESTRUTURAS DE PROGRAMAÇÃO**

### **2.1. Diagrama de bloco**

O diagrama de blocos é uma padronização que representa os passos lógicos de um determinado processamento.

Com o diagrama definimos uma sequência de símbolos com significado bem definido. Portanto, sua principal função é a de facilitar a visualização dos passos de um processamento.

#### **Simbologia**

Existem diversos símbolos em um diagrama de bloco. No quadro abaixo estão representados alguns dos símbolos mais utilizados:

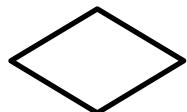
Símbolo	Função
Terminador	 Indica o início e o fim de um processamento.
Processamento	 Processamento em geral.

Entrada  
Manual



Indica a entrada de dados através do teclado.

Decisão



Indica um ponto no qual deverá ser efetuada uma escolha entre duas situações possíveis.

Exibição



Mostra os resultados obtidos com um processamento.

Documento



Indica um documento utilizado pelo processamento, seja para entrada de informações ou para exibição dos dados disponíveis após um processamento.



*Fique  
atento*

Cada símbolo irá conter uma descrição pertinente à forma com o qual o mesmo foi utilizado no fluxo, indicando o processamento ou a informação que o mesmo representa.

## Representação de algoritmos através de diagramas de bloco

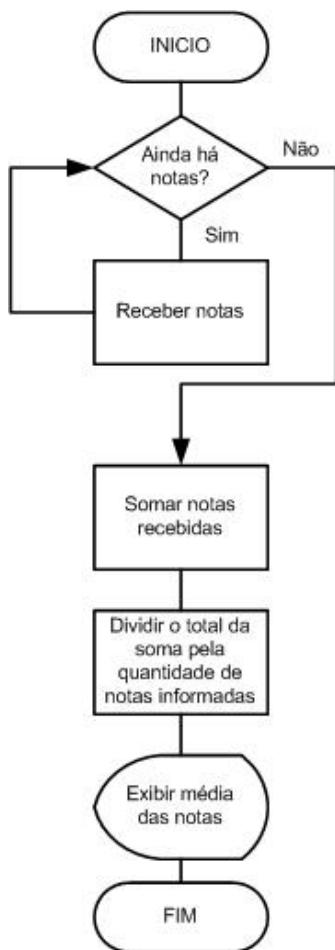
### Algoritmo 01: Fritar um ovo

1. Pegar frigideira, ovo, óleo e sal.
2. Colocar óleo na frigideira.
3. Acender o fogo.
4. Colocar a frigideira no fogo.
5. Esperar o óleo esquentar.
6. Quebrar o ovo na frigideira.
7. Jogar a casca no lixo.
8. Retirar a frigideira do fogo quando o ovo estiver no ponto.
9. Desligar o fogo.
10. Colocar sal a gosto.



Algoritmo 02: Calcular a média de notas

1. Enquanto houver notas a serem recebidas:
  - a. Receber a nota.
2. Some todas as notas recebidas.
3. Divida o total obtido pela quantidade de notas recebidas.
4. Exiba a média das notas.



## **2.2. Estruturas de decisão e repetição**

A utilização de estruturas de decisão e repetição em um algoritmo permite a realização de ações relacionadas às situações que influenciam na execução e na solução do problema.

Como foco na utilização da linguagem ADVPL, serão ilustradas as seguintes estruturas:

### Estruturas de decisão

- IF...ELSE
- DO CASE ... CASE

### Estruturas de repetição

- WHILE...END
- FOR...NEXT

Os comandos de decisão são utilizados em algoritmos cuja solução não é obtida através da utilização de ações meramente sequenciais, permitindo que estes comandos de decisão avaliem as condições necessárias para optar por uma ou outra maneira de continuar seu fluxo.

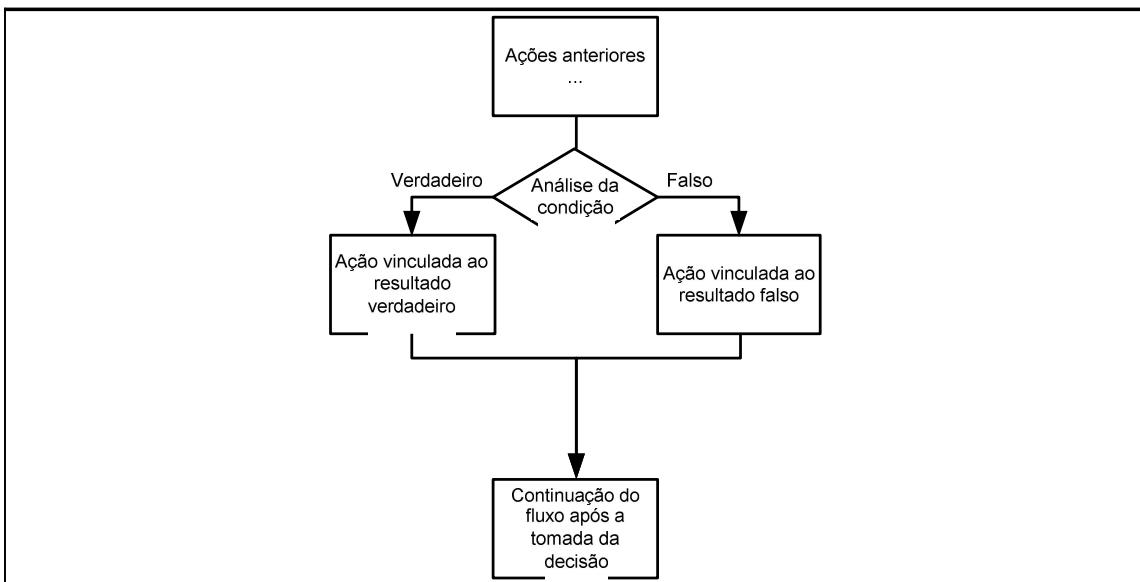
As estruturas de decisão que serão analisadas são:

IF...ELSE  
DO CASE ... CASE

### IF...ELSE

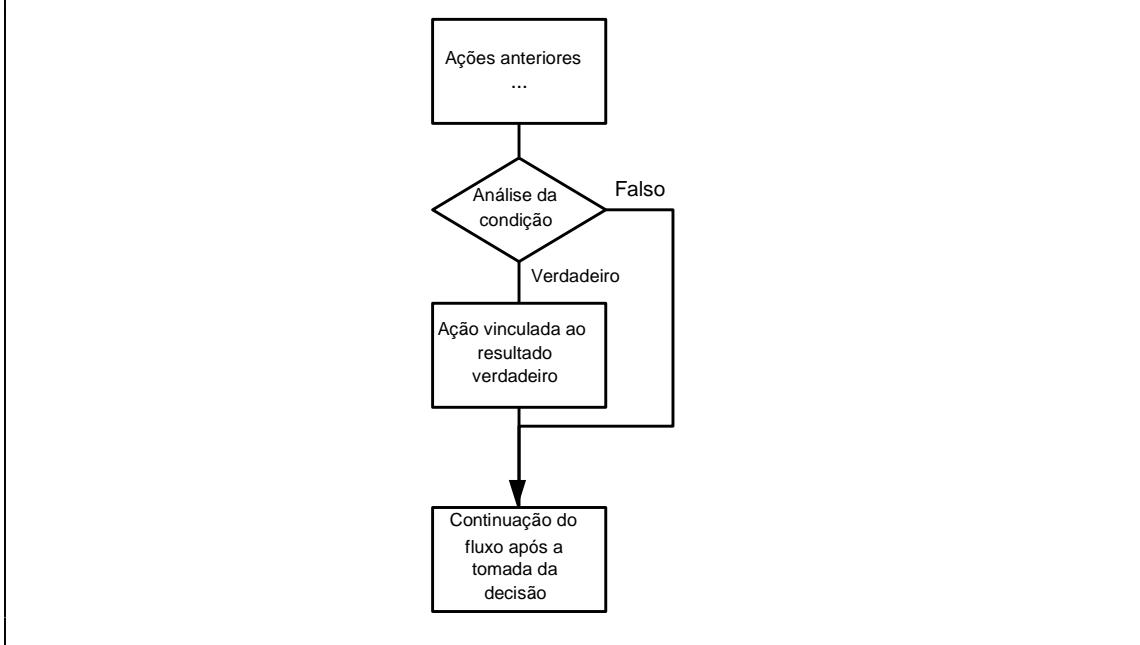
A estrutura IF...ELSE (Se/Senão) permite a análise de uma condição e a partir da qual ser executada uma de duas ações possíveis: Se a análise da condição resultar em um valor verdadeiro ou, se a análise da condição resultar em um valor falso.

Representação 01: IF...ELSE com ações para ambas as situações



Esta estrutura permite ainda que seja executada apenas uma ação, na situação em que a análise da condição resultar em um valor verdadeiro.

#### Representação 02: IF...ELSE somente com ação para situação verdadeira



*Fique  
atento*

Apesar das linguagens de programação possuírem variações para a estrutura IF...ELSE, conceitualmente todas as representações podem ser descritas com base no modelo apresentado.



**Dica**

A linguagem ADVPL possui uma variação para a estrutura IF...ELSE, descrita como IF...ELSEIF...ELSE.

Com esta estrutura é possível realizar a análise de diversas condições em sequência, para as quais será avaliada somente a ação da primeira expressão cujo análise resultar em um valor verdadeiro.



**Anotações**

---

---

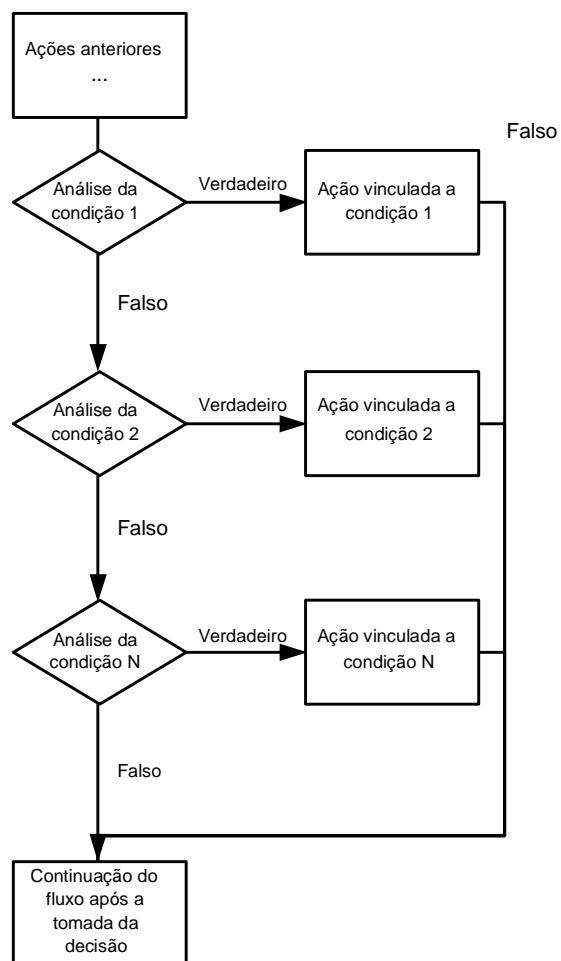
---

---

## DO CASE...CASE

A estrutura DO CASE...ENDCASE (Caso) permite a análise de diversas condições consecutivas, para as quais somente a condição para a primeira condição verdadeira será sua ação vinculada e executada.

O recurso de análise de múltiplas condições é necessário para solução de problemas mais complexos, nos quais as possibilidades de solução superam a mera análise de um único resultado verdadeiro, ou falso.





*Fique  
atento*

Apesar das linguagens de programação possuírem variações para a estrutura DO CASE...CASE, conceitualmente todas as representações podem ser descritas com base no modelo apresentado.

## 2.2.2.

### Estruturas de repetição

Os comandos de repetição são utilizados em algoritmos nas situações em que é necessário realizar uma determinada ação, ou um conjunto de ações para um número definido ou indefinido de vezes, ou ainda enquanto uma determinada condição for verdadeira.

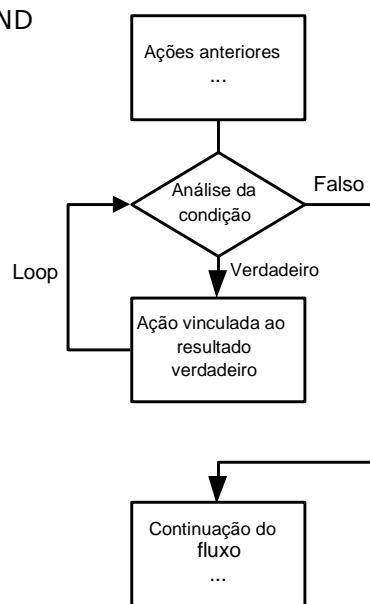
As estruturas de decisão que serão analisadas são:

WHILE...END  
FOR...TO...NEXT

WHILE...END

Nessa estrutura, o conjunto de ações é executado enquanto a análise de uma condição de referência resultar em um valor verdadeiro. É importante verificar que o bloco somente será executado, inclusive se na primeira análise a condição resultar em um valor verdadeiro

Representação: WHILE...END





*Fique  
atento*

Existem diversas variações para a estrutura WHILE...END, na qual existe a possibilidade da primeira execução ser realizada sem a análise da condição, a qual valerá apenas a partir da segunda execução.

A linguagem ADVPL aceita a sintaxe DO WHILE...ENDDO que em outras linguagens representa a situação descrita anteriormente (análise da condição somente a partir da segunda execução), mas em ADVPL esta sintaxe tem o mesmo efeito do WHILE...END.

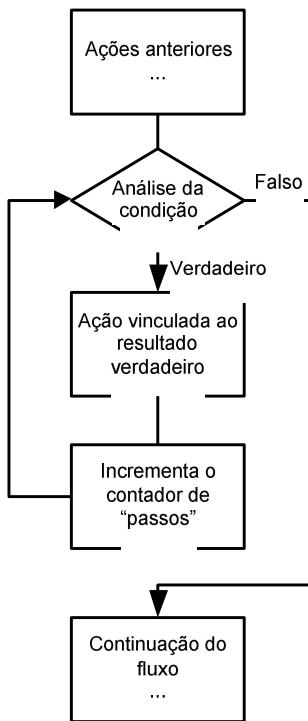
## FOR...TO...NEXT

Nessa estrutura, o conjunto de ações é executado em uma quantidade de vezes definida, normalmente referenciada como “passo”.

Para cada “passo” realizado pela estrutura FOR...TO...NEXT, é avaliada uma condição que verifica se foi atingido o número de execuções previamente definido. Dessa forma, a estrutura compreende um controle de número de “passos” executados e incrementado na análise da expressão NEXT.

Semelhante a estrutura WHILE...END, a primeira ação somente será realizada mediante um resultado verdadeiro na análise da condição.

Representação: FOR...TO...NEXT





*Fique  
atento*

A estrutura FOR...TO...NEXT, dependendo da linguagem de programação, permite a realização de um incremento simples a cada execução da instrução NEXT, ou a adição de outro valor ao contador que deverá especificado de acordo com a sintaxe da linguagem.

Em ADVPL pode ser utilizada a instrução “STEPS” para alterar o valor a ser adicionado no contador de passos a cada execução da instrução NEXT, sendo que este valor poderá ser até negativo, viabilizando uma contagem decrescente.

### **Exercícios**

#### **Exercício 02**

Vamos montar os diagramas de blocos para os algoritmos desenvolvidos no exercício anterior:

1. Usar telefone público – cartão.
2. Fritar um ovo.
3. Mascar um chiclete.
4. Trocar lâmpadas.
5. Descascar batatas.
6. Jogar o “Jogo da Forca”

## **MÓDULO 02: A LINGUAGEM ADVPL**

A Linguagem ADVPL teve seu início em 1994, sendo na verdade uma evolução na utilização de linguagens no padrão xBase pela Microsiga Software S.A. (Clipper, Visual Objects e depois FiveWin). Com a criação da tecnologia Protheus era necessário criar uma linguagem que suportasse o padrão xBase para a manutenção de todo o código existente do sistema de ERP Siga Advanced. Foi então criada a linguagem chamada Advanced Protheus Language.

O ADVPL é uma extensão do padrão xBase de comandos e funções, operadores, estruturas de controle de fluxo e palavras reservadas, contando também com funções e comandos disponibilizados pela Microsiga que a torna uma linguagem completa para a criação de aplicações ERP prontas para a Internet. Também é uma linguagem orientada a objetos e eventos, permitindo ao programador desenvolver aplicações visuais e criar suas próprias classes de objetos.

Quando compilados, todos os arquivos de código tornam-se unidades de inteligência básicas, chamados APO's (de Advanced Protheus Objects). Tais APO's são mantidos em um repositório e carregados dinamicamente pelo PROTHEUS Server para a execução. Como não existe a "linkedição", ou união física do código compilado a um determinado módulo ou aplicação, funções criadas em ADVPL podem ser executadas em qualquer ponto do Ambiente Advanced Protheus.

O compilador e o interpretador da linguagem ADVPL é o próprio servidor PROTHEUS (PROTHEUS Server), e existe um Ambiente visual para o desenvolvimento integrado (PROTHEUSIDE), em que o código pode ser criado, compilado e depurado.

Os programas em ADVPL podem conter comandos, ou funções de interface com o usuário. De acordo com tal característica, tais programas são subdivididos nas seguintes categorias:

### **Programação Com Interface Própria com o Usuário**

Nessa categoria, são classificados os programas desenvolvidos para execução através do terminal remoto do Protheus, o Protheus Remote. O Protheus Remote é a aplicação encarregada da interface e da interação com o usuário, sendo que todo o processamento do código em ADVPL, o acesso ao banco de dados e o gerenciamento de conexões é efetuado no Protheus Server. O Protheus Remote é o principal meio de acesso à execução de rotinas escritas em ADVPL no Protheus Server. Por isso permite executar qualquer tipo de código, tenha ele interface com o usuário, ou não. Porém, nessa categoria são considerados apenas os programas que realizem algum tipo de interface remota, utilizando o protocolo de comunicação do Protheus.

Poderão ser criadas rotinas para a customização do sistema ERP Microsiga Protheus, desde processos adicionais até mesmo relatórios. A grande vantagem é aproveitar todo o Ambiente montado pelos módulos do ERP Microsiga Protheus. Porém, com o ADVPL é possível até mesmo criar toda uma aplicação, ou módulo, do começo.

Todo o código do sistema ERP Microsiga Protheus é escrito em ADVPL.

## Programação Sem Interface Própria com o Usuário

---

As rotinas criadas sem interface são consideradas nesta categoria porque geralmente têm uma utilização mais específica do que um processo adicional, ou um relatório novo. Tais rotinas não têm interface com o usuário através do Protheus Remote, e qualquer tentativa nesse sentido (como a criação de uma janela padrão) ocasionará uma exceção em tempo de execução. Essas rotinas são apenas processos, ou Jobs executados no Protheus Server. Algumas vezes, a interface dessas rotinas fica a cargo de aplicações externas, desenvolvidas em outras linguagens que são responsáveis por iniciar os processos no servidor Protheus, por meio dos meios disponíveis de integração e conectividade no Protheus.

De acordo com a utilização e com o meio de conectividade utilizado, estas rotinas são subcategorizadas assim:

### Programação por Processos

Rotinas escritas em ADVPL podem ser iniciadas como processos individuais (sem interface), no Protheus Server através de duas maneiras: Iniciadas por outra rotina ADVPL por meio da chamada de funções como StartJob() ou CallProc(), ou iniciadas automaticamente, na inicialização do Protheus Server (quando propriamente configurado).

### Programação de RPC

Com a utilização de uma biblioteca de funções disponível no Protheus (uma API de comunicação) são executadas as rotinas escritas em ADVPL diretamente no Protheus Server, por aplicações externas escritas em outras linguagens. Assim, é o que se chama de RPC (de Remote Procedure Call, ou Chamada de Procedimentos Remota).

O servidor Protheus também pode executar rotinas em ADVPL, em outros servidores Protheus, através de conexão TCP/IP direta, utilizando o conceito de RPC. Do mesmo modo,

As aplicações externas podem requisitar a execução de rotinas escritas em ADVPL, pela conexão TCP/IP direta.

### Programação Web

O Protheus Server pode também ser executado como um servidor Web, respondendo a requisições HTTP. No momento dessas requisições, pode executar rotinas escritas em ADVPL como processos individuais, enviando o resultado das funções como retorno das requisições para o cliente HTTP (como por exemplo, um Browser de Internet). Qualquer rotina escrita em ADVPL que não contenha comandos de interface, pode ser executada através de requisições HTTP. O Protheus permite a compilação de arquivos HTML, contendo código ADVPL embutido. São os chamados arquivos ADVPL ASP para a criação de páginas dinâmicas.

### Programação TelNet

TelNet é parte da gama de protocolos TCP/IP que permite a conexão a um computador remoto, utilizando uma aplicação cliente desse protocolo. O PROTHEUS Server pode emular um terminal pela execução de rotinas escritas em ADVPL. Ou seja, é possível escrever rotinas ADVPL cuja interface final será um terminal TelNet, ou um coletor de dados móvel.

### **3. ESTRUTURA DE UM PROGRAMA ADVPL**

Um programa de computador nada mais é do que um grupo de comandos logicamente dispostos, com o objetivo de executar determinada tarefa. Esses comandos são gravados em um arquivo texto que é transformado em uma linguagem executável por um computador, utilizando o processo de compilação. A compilação substitui os comandos de alto nível (que os humanos comprehendem) por instruções de baixo nível (compreendida pelo sistema operacional em execução no computador). No caso do ADVPL, não é o sistema operacional de um computador que executará o código compilado, mas sim o Protheus Server.

Dentro de um programa, os comandos e funções utilizados devem seguir as regras de sintaxe da linguagem utilizada, pois caso contrário, o programa será interrompido por erros. Os erros podem ser de compilação, ou de execução.

Erros de compilação são aqueles encontrados na sintaxe que não permitem que o arquivo de código do programa seja compilado. Podem ser comandos especificados de forma errônea, utilização inválida de operadores, etc.

Erros de execução são aqueles que acontecem depois da compilação, quando o programa está sendo executado. Podem ocorrer por inúmeras razões, mas geralmente se referem às funções não existentes, ou variáveis não criadas, ou inicializadas etc.

---

#### Linhas de Programa

As linhas existentes dentro de um arquivo texto de código de programa podem ser linhas de comando, linhas de comentário, ou linhas mistas.

#### Linhas de Comando

Linhas de comando possuem os comandos, ou instruções que serão executadas. Por exemplo:

Local nCnt

Local nSoma := 0

For nCnt := 1 To 10

nSoma += nCnt

Next nCnt

## Linhas de Comentário

Linhas de comentário possuem um texto qualquer, mas não são executadas. Servem apenas para a documentação e para tornar mais fácil o entendimento do programa. Existem três formas de se comentar linhas de texto. A primeira delas é utilizar o sinal de \* (asterisco) no começo da linha:

```
* Programa para cálculo do total  
* Autor: Microsiga Software S.A.  
* Data: 2 de outubro de 2001
```

Todas as linhas iniciadas com um sinal de asterisco são consideradas como comentário. É possível utilizar a palavra NOTE, ou dois símbolos da letra "e" comercial (&&) para realizar a função do sinal de asterisco. Porém todas essas formas de comentário de linhas são obsoletas e existem apenas para a compatibilização com o padrão xBase. A melhor maneira de comentar linhas em ADVPL é utilizar duas barras transversais:

```
// Programa para cálculo do total  
// Autor: Microsiga Software S.A.  
// Data: 2 de outubro de 2001
```

Outra forma de documentar textos é utilizando as barras transversais juntamente com o asterisco. É possível comentar todo um bloco de texto sem precisar comentar linha a linha:

```
/*  
Programa para cálculo do total  
Autor: Microsiga Software S.A.  
Data: 2 de outubro de 2001  
*/
```

Todo o texto encontrado entre a abertura (indicada pelos caracteres /\*) e o fechamento (indicada pelos caracteres \*/) é considerado como comentário.

### Linhas Mistas

O ADVPL também permite que existam linhas de comando com comentário. Isto é possível adicionando-se as duas barras transversais (//) ao final da linha de comando e adicionando-se o texto do comentário:

```
Local nCnt  
Local nSoma := 0 // Inicializa a variável com zero para a soma.  
For nCnt := 1 To 10  
nSoma += nCnt  
Next nCnt
```

## Tamanho da Linha

Assim como a linha física, delimitada pela quantidade de caracteres que pode ser digitado no editor de textos utilizado, existe uma linha considerada linha lógica. A linha lógica é aquela considerada para a compilação como uma única linha de comando.

A princípio, cada linha digitada no arquivo texto é diferenciada após o pressionamento da tecla <Enter>. Ou seja, a linha lógica é a linha física no arquivo. Porém, algumas vezes por limitação física do editor de texto, ou por estética, é possível "quebrar" a linha lógica em mais de uma linha física no arquivo texto. Isso é efetuado utilizando-se o sinal de ponto-e-vírgula (;).

```
If !Empty(cNome) .And. !Empty(cEnd) .And. ; <enter>
!Empty(cTel) .And. !Empty(cFax) .And. ; <enter>
!Empty(cEmail)

GravaDados(cNome,cEnd,cTel,cFax,cEmail)

Endif
```

Nesse exemplo existe uma linha de comando para a checagem das variáveis utilizadas. Como a linha torna-se muito grande, é possível dividi-la em mais de uma linha física, utilizando o sinal de ponto-e-vírgula. Se um sinal de ponto-e-vírgula for esquecido nas duas primeiras linhas, durante a execução do programa ocorrerá um erro, pois a segunda linha física será considerada como uma segunda linha de comando na compilação. E durante a execução esta linha não terá sentido.

### 3.1. Áreas de um Programa ADVPL

Apesar de não ser uma linguagem de padrões rígidos com relação à estrutura do programa, é importante identificar algumas de suas partes. Considere o programa de exemplo abaixo:

```
#include protheus.ch

/*
+-----+
| Programa: Cálculo do Fatorial      |
| Autor   : Microsiga Software S.A.   |
| Data    : 02 de outubro de 2001       |
+-----+
*/

User Function CalcFator()

Local nCnt
Local nResultado := 1 // Resultado do fatorial
Local nFator      := 5 // Número para o cálculo

// Cálculo do fatorial
For nCnt := nFator To 1 Step -1
  nResultado *= nCnt
Next nCnt

// Exibe o resultado na tela, através da função alert
Alert("O fatorial de " + cValToChar(nFator) +
      " é " + cValToChar(nResultado))

// Termina o programa
Return
```

A estrutura de um programa ADVPL é composta pelas seguintes áreas:

**Área de Identificação**

Declaração dos includes

Declaração da função

Identificação do programa

**Área de Ajustes Iniciais**

Declaração das variáveis

**Corpo do Programa**

Preparação para o processamento

Processamento

**Área de Encerramento**

**Área de Identificação**

Esta é uma área que não é obrigatória e é dedicada a documentação do programa. Quando existente, contém apenas comentários explicando a sua finalidade, data de criação, autor, etc., e aparece no começo do programa, antes de qualquer linha de comando.

O formato para esta área não é definido. É possível escrever qualquer tipo de informação desejada e ainda, escolher a formatação apropriada.

```
#include "protheus.ch"

/*
+=====
| Programa: Cálculo do Fatorial      |
| Autor   : Microsiga Software S.A.  |
| Data    : 02 de outubro de 2001     |
+=====

*/
```

User Function CalcFator()

Opcionalmente, é possível incluir definições de constantes utilizadas no programa ou inclusão de arquivos de cabeçalho nesta área.

### Área de Ajustes Iniciais

Nessa área geralmente são feitos os ajustes iniciais, importantes para o correto funcionamento do programa. Entre os ajustes se encontram declarações de variáveis, inicializações, abertura de arquivos, etc. Apesar do ADVPL não ser uma linguagem rígida e as variáveis poderem ser declaradas em qualquer lugar do programa, é aconselhável fazê-lo nessa área. Assim, é possível tornar o código legível e facilitar a identificação de variáveis não utilizadas.

```
Local nCnt
Local nResultado := 0 // Resultado do fatorial
Local nFator      := 10 // Número para o cálculo
```

### Corpo do Programa

Nessa área encontram-se as linhas de código do programa. Nela é realiza a tarefa necessária por meio da organização lógica dessas linhas de comando. Espera-se que as linhas de comando estejam organizadas de tal modo que no final da mesma, o resultado esperado seja

obtido, seja ele armazenado em um arquivo, ou em variáveis de memória, pronto para ser exibido ao usuário através de um relatório ou na tela.

```
// Cálculo do fatorial  
For nCnt := nFator To 1 Step -1  
nResultado *= nCnt  
Next nCnt
```

A preparação para o processamento é formada pelo conjunto de validações e processamentos necessários antes da realização do processamento em si.

Avaliando o processamento do cálculo do fatorial descrito anteriormente, é possível definir que a validação inicial a ser realizada é o conteúdo da variável nFator, pois a mesma determinará a correta execução do código.

```
// Cálculo do fatorial
nFator := GetFator()
// GetFator – função ilustrativa na qual a variável recebe a informação do usuário.

If nFator <= 0
    Alert("Informação inválida")
    Return
Endif

For nCnt := nFator To 1 Step -1
    nResultado *= nCnt
Next nCnt
```

## Área de Encerramento

É nesta área onde as finalizações são efetuadas. É onde os arquivos abertos são fechados, e o resultado da execução do programa é utilizado. É possível exibir o resultado armazenado em uma variável, ou em um arquivo. Ainda, finalizar, se a tarefa já tiver sido toda completada no corpo do programa. É nessa área que se encontra o encerramento do programa. Todo programa em ADVPL deve sempre terminar com a palavra chave return.

```
// Exibe o resultado na tela, através da função alert  
Alert("O factorial de " + cValToChar(nFator) + ;  
    " é " + cValToChar(nResultado))  
  
// Termina o programa  
Return
```



*Anotações*

---

---

---

---

## **4. DECLARAÇÃO E ATRIBUIÇÃO DE VARIÁVEIS**

### **4.1. Tipo de Dados**

O ADVPL não é uma linguagem de tipos rígidos (strongly typed), o que significa que variáveis de memória podem receber diferentes tipos de dados durante a execução do programa.

As variáveis podem também conter objetos, mas os tipos primários da linguagem são:

#### **Numérico**

O ADVPL não diferencia valores inteiros de valores com ponto flutuante. Portanto, é possível criar variáveis numéricas com qualquer valor dentro do intervalo permitido. Os seguintes elementos são do tipo de dado numérico:

2
43.53
0.5
0.00001
1000000

Uma variável do tipo de dado numérico pode conter um número de dezoito dígitos, incluindo o ponto flutuante, no intervalo de 2.2250738585072014 E-308 até 1.7976931348623158 E+308.

#### **Lógico**

Valores lógicos em ADVPL são identificados através de .T. ou .Y. para verdadeiro e .F. ou .N. para falso (independentemente se os caracteres estiverem em maiúsculo, ou minúsculo).

## Caractere

---

Strings ou cadeias de caracteres são identificadas em ADVPL por blocos de texto entre aspas duplas ("") ou aspas simples (''):

"Olá mundo!"

'Esta é uma string'

"Esta é 'outra' string"

Uma variável do tipo caractere pode conter strings com no máximo 1 MB, ou seja, 1048576 caracteres.

## Data

---

O ADVPL tem um tipo de dados específico para datas. Internamente as variáveis desse tipo de dado são armazenadas como um número correspondente à data Juliana.

Variáveis do tipo de dados Data não podem ser declaradas diretamente, e sim com a utilização de funções específicas como, por exemplo, CTOD() que converte uma string para data.

## Array

---

O Array é um tipo de dado especial. É a disposição de outros elementos em colunas e linhas. O ADVPL suporta arrays unidimensionais (vetores) ou multidimensionais (matrizes). Os elementos de um array são acessados através de índices numéricos iniciados em 1, identificando a linha e coluna para quantas dimensões existirem.

Arrays devem ser utilizadas com cautela, pois se forem muito grandes podem exaurir a memória do servidor.

## Bloco de Código

---

O bloco de código é um tipo de dado especial. É utilizado para armazenar instruções escritas em ADVPL que poderão ser executadas posteriormente.

## 4.2. Declaração de variáveis

Variáveis de memória são um dos recursos mais importantes de uma linguagem. São áreas de memória criadas para armazenar informações utilizadas por um programa para a execução de tarefas. Por exemplo, quando o usuário digita uma informação qualquer, como o nome de um produto, em uma tela de um programa esta informação é armazenada em uma variável de memória para posteriormente ser gravada, ou impressa.

A partir do momento que uma variável é criada, não é mais necessário referenciar ao seu conteúdo, e sim, seu nome.

O nome de uma variável é um identificador único que deve respeitar um número máximo de 10 caracteres. O ADVPL não impede a criação de uma variável de memória cujo nome contenha mais de 10 caracteres, porém apenas os 10 primeiros serão considerados para a localização do conteúdo armazenado. Portanto, se forem criadas duas variáveis cujos 10 primeiros caracteres forem iguais, como nTotalGeralAnual e nTotalGeralMensal, as referências a qualquer uma delas no programa resultarão o mesmo, ou seja, serão a mesma variável:

```
nTotalGeralMensal := 100  
nTotalGeralAnual := 300  
Alert("Valor mensal: " + cValToChar(nTotalGeralMensal))
```

Quando o conteúdo da variável nTotalGeralMensal é exibido, o seu valor será de 300. Isso acontece porque no momento em que esse valor foi atribuído à variável nTotalGeralAnual, o ADVPL considerou apenas os 10 primeiros caracteres (assim como o faz quando deve exibir o valor da variável nTotalGeralMensal). Ou seja, considerou-as como a mesma variável. Assim o valor original de 100 foi substituído pelo de 300.

## **4.3. Escopo de variáveis**

O ADVPL não é uma linguagem de tipos rígidos para variáveis, ou seja, não é necessário informar o tipo de dados que determinada variável irá conter no momento de sua declaração, e o seu valor pode mudar durante a execução do programa.

Também não há necessidade de declarar variáveis em uma seção específica do seu código fonte, embora seja aconselhável declarar todas as variáveis necessárias no começo, tornando a manutenção mais fácil, e evitando a declaração de variáveis desnecessárias.

Para declarar uma variável é necessário utilizar um identificador de escopo. Um identificador de escopo é uma palavra chave que indica a que contexto do programa a variável declarada pertence. O contexto de variáveis pode ser local (visualizadas apenas dentro do programa atual), público (visualizadas por qualquer outro programa), entre outros.

---

### O Contexto de Variáveis dentro de um Programa

As variáveis declaradas em um programa ou função são visíveis de acordo com o escopo onde são definidas. Como também do escopo depende o tempo de existência das variáveis. A definição do escopo de uma variável é efetuada no momento de sua declaração.

Local nNúmero := 10

Esta linha de código declara uma variável chamada nNúmero, indicando que aonde pertence seu escopo é local.

Os identificadores de escopo são:

Local  
Static  
Private  
Public

O ADVPL não é rígido em relação à declaração de variáveis no começo do programa. A inclusão de um identificador de escopo não é necessário para a declaração de uma variável, contanto que um valor lhe seja atribuído.

nNumero2 := 15

Quando um valor é atribuído a uma variável em um programa ou função, o ADVPL criará a variável se ela não tiver sido declarada anteriormente. A variável então é criada como se tivesse sido declarada como Private.

Devido a essa característica, quando se pretende fazer uma atribuição a uma variável declarada previamente, mas escreve-se o nome da variável de forma incorreta, o ADVPL não gerará nenhum erro de compilação ou de execução. Assim, compreenderá o nome da variável escrito de forma incorreta como se fosse a criação de uma nova variável. Dessa forma, será alterada a lógica do programa, e é um erro muitas vezes difícil de identificar.

## Variáveis de escopo local

---

Variáveis de escopo local são pertencentes apenas ao escopo da função onde foram declaradas e devem ser explicitamente declaradas com o identificador LOCAL, como no exemplo:

```
Function Pai()
Local nVar := 10, aMatriz := {0,1,2,3}

.
<comandos>

.
Filha()

.
<mais comandos>

.
Return(T.)
```

Neste exemplo, a variável nVar foi declarada como local e atribuída com o valor 10. Quando a função Filha é executada, nVar ainda existe mas não pode ser acessada. Quando a execução da função Pai terminar, a variável nVar é destruída. Qualquer variável, com o mesmo nome no programa que chamou a função Pai, não é afetada.

Variáveis de escopo local são criadas automaticamente, cada vez que a função onde forem declaradas for ativada. Elas continuam a existir e mantêm seu valor até o fim da ativação da função (ou seja, até que a função retorne o controle para o código que a executou). Se uma função é chamada recursivamente (por exemplo, chama a si mesma), cada chamada em recursão cria um novo conjunto de variáveis locais.

A visibilidade de variáveis de escopo locais é idêntica ao escopo de sua declaração, ou seja, a variável é visível em qualquer lugar do código fonte em que foi declarada. Se uma função é chamada recursivamente, apenas as variáveis de escopo local criadas na mais recente ativação são visíveis.

## Variáveis de escopo static

Variáveis de escopo static funcionam basicamente como as variáveis de escopo local, mas mantêm seu valor através da execução e devem ser declaradas explicitamente no código, com o identificador STATIC.

O escopo das variáveis static depende de onde são declaradas. Se forem declaradas dentro do corpo de uma função ou procedimento, seu escopo será limitado àquela rotina. Se forem declaradas fora do corpo de qualquer rotina, seu escopo afeta a todas as funções declaradas no fonte.

Neste exemplo, a variável nVar é declarada como static e inicializada com o valor 10:

```
Function Pai()
Static nVar := 10
.
<comandos>
.
Filha()
.
<mais comandos>
.
Return(T.)
```

Quando a função Filha é executada, nVar ainda existe mas não pode ser acessada. Diferente de variáveis declaradas como LOCAL ou PRIVATE, nVar continua a existir e mantém seu valor atual quando a execução da função Pai termina. Entretanto, somente pode ser acessada por execuções subsequentes da função Pai.

### Variáveis de escopo private

---

A declaração é opcional para variáveis privadas. Mas podem ser declaradas explicitamente com o identificador PRIVATE.

Adicionalmente, a atribuição de valor a uma variável não criada anteriormente, de forma automática cria-se a variável como privada. Uma vez criada, uma variável privada continua a existir e mantém seu valor até que o programa ou função onde foi criada termine (ou seja, até que a função onde foi feita retorno para o código que a executou). Neste momento, é automaticamente destruída.

É possível criar uma nova variável privada com o mesmo nome de uma variável já existente. Entretanto, a nova (duplicada) variável pode apenas ser criada em um nível de ativação inferior ao nível onde a variável foi declarada pela primeira vez (ou seja, apenas em uma função chamada pela função onde a variável já havia sido criada). A nova variável privada esconderá qualquer outra variável privada ou pública (veja a documentação sobre variáveis públicas) com o mesmo nome enquanto existir.

Uma vez criada, uma variável privada é visível em todo o programa, enquanto não for destruída automaticamente. Quando a rotina que a criou terminar, ou uma outra variável privada com o mesmo nome for criada em uma subfunção chamada (neste caso, a variável existente torna-se inacessível até que a nova variável privada seja destruída).

Em termos mais simples, uma variável privada é visível dentro da função de criação e todas as funções chamadas por esta, a menos que uma função chamada crie sua própria variável privada com o mesmo nome.

Por exemplo:

```
Function Pai()
Private nVar := 10
<comandos>
.
Filha()
<mais comandos>
.
Return(T.)
```

Neste exemplo, a variável nVar é criada com escopo private e inicializada com o valor 10. Quando a função Filha é executada, nVar ainda existe e, diferente de uma variável de escopo local, pode ser acessada pela função Filha. Quando a função Pai terminar, nVar será destruída e qualquer declaração de nVar anterior se tornará acessível novamente.



*Fique  
atento*

No ambiente ERP Protheus existe uma convenção adicional que fala sobre as variáveis, em uso pela aplicação, não sejam incorretamente manipuladas. Por esta convenção deve ser adicionado o caractere “\_” antes do nome das variáveis PRIVATE e PUBLIC. Para maiores informações avaliar o tópico: “Boas Práticas de Programação”.

Exemplo: Private \_dData

## Variáveis de escopo public

---

É possível criar variáveis de escopo public dinamicamente, no código com o identificador PUBLIC. As variáveis desse escopo continuam a existir e mantêm seu valor até o fim da execução da thread (conexão). Também é possível criar uma variável de escopo private com o mesmo nome de uma variável de escopo public existente, entretanto, não é permitido criar uma variável de escopo public com o mesmo nome de uma variável de escopo private existente.

Uma vez criada, uma variável de escopo public é visível em todo o programa em que foi declarada, até que seja escondida por uma variável de escopo private, criada com o mesmo nome. A nova variável de escopo private criada esconde a variável de escopo public existente, e esta se tornará inacessível até que a nova variável private seja destruída. Por exemplo:

```
Function Pai()
Public nVar := 10
<comandos>
.
Filha()
<mais comandos>
.
Return(.T.)
```

Nesse exemplo, nVar é criada como public e inicializada com o valor 10. Quando a função Filha é executada, nVar ainda existe e pode ser acessada. Diferente de variáveis locais ou privates, nVar ainda existe após o término da a execução da função Pai.

Diferentemente dos outros identificadores de escopo, quando uma variável é declarada como pública sem ser inicializada, o valor assumido é falso (.F.) e não nulo (nil).



*Fique  
atento*

No ambiente ERP Protheus, existe uma convenção adicional que deve ser respeitada. Essa convenção diz que as variáveis em uso pela aplicação não sejam incorretamente manipuladas. Por essa convenção deve ser adicionado o caractere “\_” antes do nome de variáveis PRIVATE e PUBLIC. Para Maiores informações avalie o tópico: Boas Práticas de Programação.

Exemplo: Public \_cRotina

#### **4.4. Entendendo a influência do escopo das variáveis**

Considere as linhas de código de exemplo:

```
nResultado := 250 * (1 + (nPercentual / 100))
```

Se esta linha for executada em um programa ADVPL ocorrerá um erro de execução com a mensagem "variable does not exist: nPercentual", pois esta variável está sendo utilizada em uma expressão de cálculo sem ter sido declarada. Para solucionar esse erro, deve-se declarar a variável previamente:

```
Local nPercentual, nResultado  
nResultado := 250 * (1 + (nPercentual / 100))
```

No exemplo acima, as variáveis são declaradas previamente, utilizando o identificador de escopo local. Quando a linha de cálculo for executada, o erro de variável não existente não mais ocorrerá. Porém as variáveis não inicializadas têm sempre o valor default nulo (Nil) e este valor não pode ser utilizado em um cálculo, pois também gerará erros de execução (nulo não pode ser dividido por 100). A resolução deste problema é efetuada inicializando-se a variável através de uma das formas:

```
Local nPercentual, nResultado  
nPercentual := 10  
nResultado := 250 * (1 + (nPercentual / 100))  
ou  
Local nPercentual := 10, nResultado  
nResultado := 250 * (1 + (nPercentual / 100))
```

A diferença, entre o último exemplo e os dois anteriores, é que a variável é inicializada no momento da declaração. Nos dois exemplos, a variável é primeiro declarada e então inicializada em uma outra linha de código.

É aconselhável optar pelo operador de atribuição composto de dois pontos e sinal de igual, pois o operador de atribuição, utilizando somente o sinal de igual, pode ser facilmente confundido com o operador relacional (para comparação), durante a criação do programa.

## **4.5. Operações com Variáveis**

Uma vez que um valor lhe seja atribuído, o tipo de dado de uma variável é igual ao tipo de dado do valor atribuído. Ou seja, uma variável passa a ser numérica se um número lhe é atribuído, passa a ser caractere se uma string de texto lhe for atribuída etc. Porém, mesmo que uma variável seja de determinado tipo de dado, é possível mudar o tipo de variável atribuindo outro tipo a ela:

```
01 Local xVariavel // Declara a variável inicialmente com valor nulo
02
03 xVariavel := "Agora a variável é caractere..."
04 Alert("Valor do Texto: " + xVariavel)
05
06 xVariavel := 22 // Agora a variável é numérica
07 Alert(cValToChar(xVariavel))
08
09 xVariavel := .T. // Agora a variável é lógica
10 If xVariavel
11     Alert("A variável tem valor verdadeiro...")
12 Else
13     Alert("A variável tem valor falso...")
14 Endif
15
16 xVariavel := Date() // Agora a variável é data
17 Alert("Hoje é: " + DtoC(xVariavel))
18
19 xVariavel := nil // Nulo novamente
20 Alert("Valor nulo: " + xVariavel)
21
22 Return
```

No programa de exemplo anterior, a variável xVariavel é utilizada para armazenar diversos tipos de dados. A letra "x", em minúsculo no começo do nome, é utilizada para indicar uma variável que pode conter diversos tipos de dados, segundo a Notação Húngara (consulte documentação específica para detalhes). Este programa troca os valores da variável e exibe seu conteúdo para o usuário através da função ALERT(). Essa função recebe um parâmetro que deve ser do tipo string de caractere. Por isso, dependendo do tipo de dado da variável xVariavel, é necessário fazer uma conversão antes.

Apesar dessa flexibilidade de utilização de variáveis, devem-se tomar cuidados na passagem de parâmetros para funções ou comandos, e na concatenação (ou soma) de valores. Note a linha 20 do programa de exemplo. Quando esta linha é executada, a variável xVariavel contém o valor nulo. A tentativa de soma de tipos de dados diferentes gera erro de execução do programa. Nesta linha do exemplo, ocorrerá um erro com a mensagem "type mismatch on +".

Excetuando-se o caso do valor nulo, para os demais devem ser utilizadas funções de conversão, quando é necessário concatenar tipos de dados diferentes (por exemplo, nas linhas 07 e 17).

Note também que quando uma variável é do tipo de dado lógico, ela pode ser utilizada diretamente para checagem (linha 10):

```
If xVariavel  
    é o mesmo que  
If xVariavel = .T.
```

#### 4.5.2.

#### Operadores da linguagem ADVPL

##### Operadores comuns

Na documentação sobre variáveis há uma breve demonstração de como atribuir valores a uma variável da forma mais simples. O ADVPL amplia significativamente a utilização de variáveis, através do uso de expressões e funções.

Uma expressão é um conjunto de operadores e operandos, cujo resultado pode ser atribuído a uma variável ou então analisado para a tomada de decisões. Por exemplo:

```
Local nSalario := 1000, nDesconto := 0.10
Local nAumento, nSalLiquido
nAumento := nSalario * 1.20
nSalLiquido := nAumento * (1-nDesconto)
```

Nesse exemplo são utilizadas algumas expressões para calcular o salário líquido após um aumento. Os operandos de uma expressão podem ser uma variável, uma constante, um campo de arquivo, ou uma função.

## Operadores Matemáticos

Os operadores utilizados em ADVPL para cálculos matemáticos são:

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
** ou ^	Exponenciação
%	Módulo (Resto da Divisão)

## Operadores de String

Os operadores utilizados em ADVPL para tratamento de caracteres são:

+	Concatenação de strings (união).
-	Concatenação de strings com eliminação dos brancos finais das strings intermediárias.
\$	Comparação de Substrings (contido em).

## Operadores Relacionais

---

Os operadores utilizados em ADVPL para operações e avaliações relacionais são:

<	Comparação Menor
>	Comparação Maior
=	Comparação Igual
==	Comparação Exatamente Igual (para caracteres)
<=	Comparação Menor ou Igual
>=	Comparação Maior ou Igual
<> ou # ou !=	Comparação Diferente

## Operadores Lógicos

---

Os operadores utilizados em ADVPL para operações e avaliações lógicas são:

.And.	E lógico
.Or.	OU lógico
.Not. ou !	NÃO lógico

## Operadores de Atribuição

Os operadores utilizados em ADVPL para atribuição de valores a variáveis de memória são:

<code>:=</code>	Atribuição Simples
<code>+≡</code>	Adição e Atribuição em Linha
<code>-≡</code>	Subtração e Atribuição em Linha
<code>*≡</code>	Multiplicação e Atribuição em Linha
<code>/≡</code>	Divisão e Atribuição em Linha
<code>**= ou ^≡</code>	Exponenciação e Atribuição em Linha
<code>%≡</code>	Módulo (resto da divisão) e Atribuição em Linha

Atribuição Simples

O sinal de igualdade é utilizado para atribuir valor a uma variável de memória.

`nVariavel := 10`

## Atribuição em Linha

O operador de atribuição em linha é caracterizado por dois pontos e o sinal de igualdade. Tem a mesma função do sinal de igualdade sozinho, porém aplica a atribuição às variáveis. Com ele é possível atribuir mais de uma variável ao mesmo tempo.

```
nVar1 := nVar2 := nVar3 := 0
```

Quando diversas variáveis são inicializadas em uma mesma linha, a atribuição começa da direita para a esquerda, ou seja, nVar3 recebe o valor zero inicialmente, nVar2 recebe o conteúdo de nVar3 e nVar1 recebe o conteúdo de nVar2 por final.

Com o operador de atribuição em linha, é possível substituir as inicializações individuais de cada variável por uma inicialização apenas:

```
Local nVar1 := 0, nVar2 := 0, nVar3 := 0
```

por

```
Local nVar1 := nVar2 := nVar3 := 0
```

O operador de atribuição em linha também pode ser utilizado para substituir valores de campos em um banco de dados.

## Atribuição Composta

Os operadores de atribuição composta são uma facilidade da linguagem ADVPL para expressões de cálculo e atribuição. Dessa forma é possível economizar digitação:

Operador	Exemplo	Equivalente a
<code>+=</code>	<code>X += Y</code>	<code>X = X + Y</code>
<code>-=</code>	<code>X -= Y</code>	<code>X = X - Y</code>
<code>*=</code>	<code>X *= Y</code>	<code>X = X * Y</code>
<code>/=</code>	<code>X /= Y</code>	<code>X = X / Y</code>
<code>**= ou ^=</code>	<code>X **= Y</code>	<code>X = X ** Y</code>
<code>%=</code>	<code>X %= Y</code>	<code>X = X % Y</code>

## Operadores de Incremento/Decremento

---

A linguagem ADVPL possui operadores para realizar incremento ou decremeno de variáveis. Entende-se por incremento aumentar o valor de uma variável numérica em 1 e entende-se por decremeno diminuir o valor da variável em 1. Os operadores são:

<code>++</code>	Incremento Pós ou Pré-fixado
<code>--</code>	Decremento Pós ou Pré-fixado

Os operadores de decremeno/incremento podem ser colocados tanto antes (pré-fixado) como depois (pós-fixado) do nome da variável. Dentro de uma expressão, a ordem do operador é muito importante, podendo alterar o resultado da expressão. Os operadores incrementais são executados da esquerda para a direita dentro de uma expressão.

```
Local nA := 10
Local nB := nA++ + nA
```

O valor da variável nB resulta em 21, pois a primeira referência a nA (antes do++) continha o valor 10 que foi considerado e imediatamente aumentado em 1. Na segunda referência a nA, este já possuía o valor 11. O que foi efetuado foi a soma de 10 mais 11, igual a 21. O resultado final após a execução dessas duas linhas é a variável nB contendo 21 e a variável nA contendo 11.

No entanto:

Local nA := 10
Local nB := ++nA + nA

Resulta em 22 pois o operador incremental aumentou o valor da primeira nA antes que seu valor fosse considerado.

## Operadores Especiais

Além dos operadores comuns, o ADVPL possui alguns outros operadores, ou identificadores. Estas são suas finalidades:

()	Agrupamento ou Função
[]	Elemento de Matriz
{}	Definição de Matriz, Constante ou Bloco de Código
->	Identificador de Apelido
&	Macro substituição
@	Passagem de parâmetro por referência
	Passagem de parâmetro por valor

Os parênteses são utilizados para agrupar elementos em uma expressão, mudando a ordem de precedência da avaliação da expressão (segundo as regras matemáticas, por exemplo). Também servem para envolver os argumentos de uma função.

Os colchetes são utilizados para especificar um elemento específico de uma matriz. Por exemplo, A[3,2] refere-se ao elemento da matriz A na linha 3, coluna 2.

As chaves são utilizadas para a especificação de matrizes literais, ou blocos de código. Por exemplo, A:={10,20,30} cria uma matriz chamada A com três elementos.

O símbolo -> identifica um campo de um arquivo, diferenciando-o de uma variável. Por exemplo, FUNC->nome refere-se ao campo nome do arquivo FUNC. Mesmo que exista uma variável chamada nome, é o campo nome que será acessado.

O símbolo & identifica uma avaliação de expressão através de macro e é visto em detalhes na documentação sobre [macro substituição](#).

O símbolo @ é utilizado para indicar que durante a passagem de uma variável para uma função, ou procedimento ela seja tomada como uma referência e não como valor.

O símbolo || é utilizado para indicar que durante a passagem de uma variável para uma função ou procedimento, ela seja tomada como um e valor não como referência.

## **Ordem de Precedência dos Operadores**

Dependendo do tipo de operador, existe uma ordem de precedência para a avaliação dos operandos. Em princípio, todas as operações, com os operadores, são realizadas da esquerda para a direita, se eles tiverem o mesmo nível de prioridade.

A ordem de precedência, ou nível de prioridade de execução dos operadores em ADVPL é:

1. Operadores de Incremento/Decremento pré-fixado
2. Operadores de String
3. Operadores Matemáticos
4. Operadores Relacionais
5. Operadores Lógicos
6. Operadores de Atribuição
7. Operadores de Incremento/Decremento pós-fixado

Em expressões complexas com diferentes tipos de operadores, a avaliação seguirá essa sequência. Caso exista mais de um operador do mesmo tipo (ou seja, de mesmo nível), a avaliação se dá da esquerda para direita. Para os operadores matemáticos entretanto, há uma precedência a seguir:

1. Exponenciação
2. Multiplicação e Divisão
3. Adição e Subtração

Considere o exemplo:

```
Local nResultado := 2+10/2+5*3+2^3
```

O resultado desta expressão é 30, pois primeiramente é calculada a exponenciação  $2^3(=8)$ , então são calculadas as multiplicações e divisões  $10/2(=5)$  e  $5*3(=15)$ , e finalmente as adições resultando em  $2+5+15+8(=30)$ .

### **Alteração da Precedência**

A utilização de parênteses dentro de uma expressão altera a ordem de precedência dos operadores. Operandos entre parênteses são analisados antes dos que se encontram

fora dos parênteses. Se existirem mais de um conjunto de parênteses não-aninhados, o grupo mais a esquerda será avaliado primeiro e assim sucessivamente.

```
Local nResultado := (2+10)/(2+5)*3+2^3
```

No exemplo acima, primeiro será calculada a exponenciação  $2^3 (=8)$ . Em seguida  $2+10 (=12)$  será calculado,  $2+5 (=7)$  calculado, e finalmente a divisão e a multiplicação serão efetuadas, o que resulta em  $12/7*3+8 (=13.14)$ .

Se existirem vários parênteses aninhados, ou seja, colocados um dentro do outro, a avaliação ocorrerá do parênteses mais interno em direção ao mais externo.

#### 4.5.3.

#### Operação de Macro Substituição

O operador de macro substituição, simbolizado pelo "e" comercial (&), é utilizado para a avaliação de expressões em tempo de execução. Funciona como se uma expressão armazenada fosse compilada em tempo de execução, antes de ser de fato executada.

Considere o exemplo:

```
01 X := 10
02 Y := "X + 1"
03 B := &Y // O conteúdo de B será 11
```

A variável X é atribuída com o valor 10, enquanto a variável Y é atribuída com a string de caracteres contendo "X + 1".

A terceira linha utiliza o operador de macro. Esta linha faz com que o número 11 seja atribuído à variável B. É possível perceber que esse é o valor resultante da expressão em formato de caractere contida na variável Y.

Utilizando-se uma técnica matemática elementar, a substituição, temos que na segunda linha, Y é definido como "X + 1", então é possível substituir Y na terceira linha:

```
03 B := &"X + 1"
O operador de macro cancela as aspas:
03 B := X + 1
```

O operador de macro remove as aspas, o que deixa uma parte do código para ser executado.

Deve-se ter em mente que tudo isso acontece em tempo de execução, o que torna tudo muito dinâmico. Uma utilização interessante é criar um tipo de calculadora ou, um avaliador de fórmulas, que determina o resultado de algo que o usuário digita.

O operador de macro tem uma limitação: variáveis referenciadas dentro da string de caracteres (X nos exemplos anteriores) não podem ser locais.



Anotações

---

---

---

---

#### 4.5.4.

#### Funções de manipulação de variáveis

Além de atribuir, controlar o escopo e a macro executar o conteúdo das variáveis, é necessário manipular seu conteúdo através de funções específicas da linguagem para cada situação.

As operações de manipulação de conteúdo mais comuns em programação são:

- Conversões entre tipos de variáveis.
- Manipulação de strings.
- Manipulação de variáveis numéricas.
- Verificação de tipos de variáveis.
- Manipulação de arrays.
- Execução de blocos de código.

Neste tópico serão abordadas as conversões entre tipos de variáveis e as funções de manipulação de strings e variáveis numéricas.

##### Conversões entre tipos de variáveis

As funções mais utilizadas nas operações entre conversão entre tipos de variáveis são:

- CTOD()
- CVALTOCHAR()
- DTOC()
- DTOS()
- STOD()
- STR()
- STRZERO()
- VAL()

## CTOD()

Sintaxe	CTOD(cData)
Descrição	Realiza a conversão de uma informação do tipo caractere no formato “DD/MM/AAAA”, para uma variável do tipo data.

## CVALTOCHAR()

Sintaxe	CVALTOCHAR(nValor)
Descrição	Realiza a conversão de uma informação do tipo numérico em uma string, sem a adição de espaços a informação.

## DTOC()

Sintaxe	DTOC(dData)
Descrição	Realiza a conversão de uma informação do tipo data para em caractere, sendo o resultado no formato “DD/MM/AAAA”.

## DTOS()

Sintaxe	DTOS(dData)
Descrição	Realiza a conversão de uma informação do tipo data em um caractere, sendo o resultado no formato “AAAAMMDD”.

## STOD()

Sintaxe	STOD(sData)
Descrição	Realiza a conversão de uma informação do tipo caractere, com conteúdo no formato “AAAAMMDD”, em data.

## STR()

Sintaxe	STR(nValor)
Descrição	Realiza a conversão de uma informação do tipo numérico em uma string, adicionando espaços à direita.

## STRZERO()

Sintaxe	STRZERO(nValor, nTamanho)
Descrição	Realiza a conversão de uma informação do tipo numérico em uma string, adicionando zeros à esquerda do número convertido, de forma que a string gerada tenha o tamanho especificado no parâmetro.

## VAL()

Sintaxe	VAL(cValor)
Descrição	Realiza a conversão de uma informação do tipo caractere em numérica.

## Manipulação de strings

As funções mais utilizadas nas operações de manipulação do conteúdo de strings são:

ALLTRIM()

ASC()

AT()

CHR()

CSTUFF()

LEN()

RAT()

SUBSTR()

## ALLTRIM()

Sintaxe	ALLTRIM(cString)
Descrição	<p>Retorna uma string sem os espaços à direita e à esquerda, referente ao conteúdo informado como parâmetro.</p> <p>A função ALLTRIM() implementa as ações as funções RTRIM (“right trim”) e LTRIM (“left trim”).</p>

## ASC()

Sintaxe	ASC(cCaractere)
Descrição	Converte uma informação caractere em seu valor, de acordo com a tabela ASCII.

## AT()

Sintaxe	AT(cCaractere, cString )
Descrição	Retorna a primeira posição de um caractere ou string, dentro de outra string especificada.

## CHR()

Sintaxe	CHR(nASCII)
Descrição	Converte um valor número referente a uma informação da tabela ASCII, no caractere que esta informação representa.

## LEN()

Sintaxe	LEN(cString)
Descrição	Retorna o tamanho da string especificada no parâmetro.

## LOWER()

Sintaxe	LOWER(cString)
Descrição	Retorna uma string com todos os caracteres minúsculos, tendo como base a string passada como parâmetro.

## RAT()

Sintaxe	RAT(cCaractere, cString)
Descrição	Retorna a última posição de um caractere ou string, dentro de outra string especificada.

## **STUFF()**

---

Sintaxe	STUFF(cString, nPosInicial, nExcluir, cAdicao)
Descrição	Permite substituir um conteúdo caractere em uma string já existente, especificando a posição inicial para esta adição e o número de caracteres que serão substituídos.

## **SUBSTR()**

---

Sintaxe	SUBSTR(cString, nPosInicial, nCaracteres)
Descrição	Retorna parte do conteúdo de uma string especificada, de acordo com a posição inicial deste conteúdo na string e a quantidade de caracteres que deverá ser retornada a partir daquele ponto (inclusive).

## **UPPER()**

---

Sintaxe	UPPER(cString)
Descrição	Retorna uma string com todos os caracteres maiúsculos, tendo como base a string passada como parâmetro.

## **Manipulação de variáveis numéricas**

---

As funções mais utilizadas nas operações de manipulação do conteúdo de strings são:

**ABS()**

**INT()**

**NOROUND()**

**ROUND()**

## ABS()

Sintaxe	ABS(nValor)
Descrição	Retorna um valor absoluto (independente do sinal) com base no valor especificado no parâmetro.

## INT()

Sintaxe	INT(nValor)
Descrição	Retorna a parte inteira de um valor especificado no parâmetro.

## NOROUND()

Sintaxe	NOROUND(nValor, nCasas)
Descrição	Retorna um valor, truncando a parte decimal do valor especificado no parâmetro de acordo com a quantidade de casas decimais solicitadas.

## ROUND()

Sintaxe	ROUND(nValor, nCasas)
Descrição	Retorna um valor, arredondando a parte decimal do valor especificado no parâmetro, de acordo com as quantidades de casas decimais solicitadas, utilizando o critério matemático.

## Verificação de tipos de variáveis

As funções de verificação permitem a consulta ao tipo do conteúdo da variável, durante a execução do programa.

TYPE()

VALTYPE()

TYPE()

---

Sintaxe	TYPE("cVariavel")
Descrição	Determina o tipo do conteúdo de uma variável que não foi definida na função em execução.

VALTYPE()

---

Sintaxe	VALTYPE(cVariável)
Descrição	Determina o tipo do conteúdo de uma variável que foi definida na função em execução.



Anotações

---

---

---

---

## **5. ESTRUTURAS BÁSICAS DE PROGRAMAÇÃO**

O ADVPL suporta várias estruturas de controle que permitem mudar a sequência de fluxo de execução de um programa. Essas estruturas permitem a execução de código baseado em condições lógicas e a repetição da execução de pedaços de código em qualquer número de vezes.

Em ADVPL, todas as estruturas de controle podem ser "aninhadas" dentro de todas as demais estruturas, contanto que estejam aninhadas propriamente. Estruturas de controle têm um identificador de início e um de fim, e qualquer estrutura aninhada deve se encontrar entre estes identificadores.

Também existem estruturas de controle para determinar que elementos, comandos, etc. em um programa serão compilados. Estas são as diretivas do pré-processador `#ifdef...#endif` e `#ifndef...#endif`. Consulte a documentação sobre o pré-processador para maiores detalhes.

As estruturas de controle em ADVPL estão divididas em:

- Estruturas de repetição.
- Estruturas de decisão.

### **5.1. Estruturas de repetição**

Estruturas de repetição são designadas para executar uma seção de código mais de uma vez. Por exemplo, imaginando-se a existência de uma função para imprimir um relatório, digamos que é preciso imprimi-lo quatro vezes. É possível simplesmente chamar a função de impressão quatro vezes em sequência, mas isto se tornaria pouco profissional e não resolveria o problema se o número de relatórios fosse variável.

Em ADVPL existem dois comandos para a repetição de seções de código, que são os comandos [FOR...NEXT](#) e o comando [WHILE...ENDDO](#).

O Comando FOR...NEXT

A estrutura de controle FOR...NEXT, ou simplesmente o loop FOR, repete uma seção de código em um número determinado de vezes.



## Sintaxe

```
FOR Variavel := nValorInicial TO nValorFinal [STEP nIncremento]
```

Comandos...

[EXIT]

[LOOP]

NEXT

## Parâmetros

Variável	Especifica uma variável, ou um elemento de uma matriz para atuar como um contador. A variável, ou o elemento da matriz não precisa ter sido declarado antes da execução do comando FOR...NEXT. Se a variável não existir, será criada como uma <u>variável privada</u> .
nValorInicial TO nValorFinal	nValorInicial é o valor inicial para o contador; nValorFinal é o valor final para o contador. É possível utilizar valores numéricos literais, variáveis ou expressões, contanto que o resultado seja do tipo dado numérico.
STEP nIncremento	nIncremento é a quantidade que será incrementada, ou decrementada no contador após cada execução da seção de comandos. Se o valor de nIncremento for negativo, o contador será decrementado. Se a cláusula STEP for omitida, o contador será incrementado em 1. É possível utilizar os valores numéricos literais, variáveis ou expressões, contanto que o resultado seja do tipo de dado numérico.
Comandos	Especifica uma, ou mais instruções de comando ADVPL que serão executadas.
EXIT	Transfere o controle de dentro do comando FOR...NEXT para o comando imediatamente seguinte ao NEXT. Ou seja, finaliza a repetição da seção de comandos imediatamente. É possível adicionar o comando EXIT em qualquer lugar entre o FOR e o NEXT.
LOOP	Retorna o controle diretamente para a cláusula FOR sem executar o restante dos comandos entre o LOOP e o NEXT. O contador é incrementado ou decrementado normalmente, como se o NEXT tivesse sido alcançado. É possível posicionar o comando LOOP em qualquer lugar, entre o FOR e o NEXT.



*Fique  
atento*

Uma variável, ou um elemento de uma matriz é utilizado como um contador para especificar quantas vezes os comandos ADVPL dentro da estrutura FOR...NEXT são executados.

Os comandos ADVPL depois do FOR são executados até que o NEXT seja alcançado. O contador (Variável) é então incrementado, ou decrementado com o valor em nIncremento (se a cláusula STEP for omitida, o contador é incrementado em 1). Então, o contador é comparado com o valor em nValorFinal. Se for menor ou igual ao valor em nValorFinal, os comandos seguintes ao FOR são executados novamente.

Se o valor for maior que o contido em nValorFinal, a estrutura FOR...NEXT é terminada e o programa continua a execução no primeiro comando após o NEXT.

Os valores de nValorInicial, nValorFinal e nIncremento são apenas considerados inicialmente. Entretanto, mudar o valor da variável utilizada como contador dentro da estrutura afetará o número de vezes que a repetição será executada. Se o valor de nIncremento é negativo e o valor de nValorInicial é maior que o de nValorFinal, o contador será decrementado a cada repetição.

Exemplo:

Local nCnt

Local nSomaPar := 0

For nCnt := 0 To 100 Step 2

nSomaPar += nCnt

Next

Alert( "A soma dos 100 primeiros números pares é: " + ;

cValToChar(nSomaPar) )

Return

Esse exemplo imprime a soma dos 100 primeiros números pares. A soma é obtida através da repetição do cálculo, utilizando a própria variável de contador. Como a cláusula STEP está sendo utilizada, a variável nCnt será sempre incrementada em 2. E como o contador começa com 0, seu valor sempre será um número par.

## O Comando WHILE...ENDDO

---

A estrutura de controle WHILE...ENDDO, ou simplesmente o loop WHILE, repete uma seção de código enquanto uma determinada expressão resultar em verdadeiro (.T.).

### Sintaxe

```
WHILE lExpressao
  Comandos...
  [EXIT]
  [LOOP]
ENDDO
```

### Parâmetros

lExpressao	Especifica uma expressão lógica cujo valor determina quando os comandos entre o WHILE e o ENDDO são executados. Enquanto o resultado de lExpressao for avaliado como verdadeiro (.T.), o conjunto de comandos são executados.
Comandos	Especifica uma ou mais instruções de comando ADVPL, que serão executadas enquanto lExpressao for avaliado como verdadeiro (.T.).
EXIT	Transfere o controle de dentro do comando WHILE...ENDDO para o comando imediatamente seguinte ao ENDDO, ou seja, finaliza a repetição da seção de comandos imediatamente. É possível posicionar o comando EXIT em qualquer lugar entre o WHILE e o ENDO.
LOOP	Retorna o controle diretamente para a cláusula WHILE sem executar o restante dos comandos entre o LOOP e o ENDDO. A expressão em lExpressao é reavaliada para a decisão se os comandos continuarem sendo executados.

Exemplo :

```
Local nNumber := nAux := 350
nAux := Int(nAux / 2)
While nAux > 0
  nSomaPar += nCnt
  Next
  Alert( "A soma dos 100 primeiros números pares é: " +
    cValToChar(nSomaPar) )
Return
```



*Fique  
atento*

Os comandos entre o WHILE e o ENDDO são executados enquanto o resultado da avaliação da expressão em lExpressao permanecer verdadeiro (.T.). Cada palavra chave WHILE deve ter uma palavra chave ENDDO correspondente.

### 5.1.1.

### Influenciando o fluxo de repetição

A linguagem ADVPL permite a utilização de comandos que influem diretamente em um processo de repetição, sendo eles:

LOOP

EXIT

LOOP

A instrução LOOP é utilizada para forçar um desvio no fluxo do programa de volta a análise da condição de repetição. Dessa forma, todas as operações, que seriam realizadas dentro da estrutura de repetição após o LOOP, serão desconsideradas.

Exemplo:

```
aItens:= ListaProdutos() // função ilustrativa que retorna um array com dados dos produtos.
```

```
nQuantidade := Len(aItens)
```

```
nItens := 0
```

```
While nItens < nQuantidade
```

```
    nItens++
```

```
    IF BLOQUEADO(aItens [nItens]) // função ilustrativa que verifica se o produto está
```

```
        LOOP // bloqueado.
```

```
    ENDIF
```

```
    IMPRIME() // função ilustrativa que realiza a impressão de um item liberado para uso.
```

```
End
```

```
// Se o produto estiver bloqueado, o mesmo não será impresso, pois a execução da //  
instrução LOOP fará o fluxo do programa retornar a partir da análise da condição.
```

## EXIT

---

A instrução EXIT é utilizada para forçar o término de uma estrutura de repetição. Dessa forma, todas as operações que seriam realizadas dentro da estrutura de repetição após o EXIT serão desconsideradas, e o programa continuará a execução a partir da próxima instrução posterior ao término da estrutura (END ou NEXT).

Exemplo:

While .T.

```
IF MSGYESNO("Deseja jogar o jogo da forca?")
```

```
    JFORCA() // Função ilustrativa que implementa o algoritmo do jogo da forca.
```

```
ELSE
```

```
    EXIT
```

```
ENDIF
```

```
End
```

```
MSGINFO("Final de Jogo")
```

```
// Enquanto não for respondido "Não" para a pergunta: "Deseja jogar o jogo da  
// forca", será executada a função do jogo da forca.
```

```
// Caso seja selecionada a opção "Não", será executada a instrução EXIT que provocará  
o término do LOOP, permitindo a execução da mensagem de "Final de Jogo".
```



Anotações

---

---

---

---

## **5.2. Estruturas de decisão**

As Estruturas de desvio são designadas para executar uma seção de código se determinada condição lógica resultar em verdadeiro (.T.).

Em ADVPL existem dois comandos para execução de seções de código, de acordo com as avaliações lógicas, que são os comandos IF...ELSE...ENDIF e o comando DO CASE...ENDCASE.

O Comando IF...ELSE...ENDIF

Executa um conjunto de comandos baseado no valor de uma expressão lógica.

Sintaxe

```
IF lExpressao  
Comandos  
[ELSE  
Comandos...]  
ENDIF
```

Parâmetros

LExpressao	Especifica uma expressão lógica que é avaliada. Se lExpressao resultar em verdadeiro (.T.), qualquer comando seguinte ao IF e antecedente ao ELSE ou ENDIF (o que ocorrer primeiro) será executado.  Se lExpressao resultar em falso (.F.) e a cláusula ELSE for definida, qualquer comando após essa cláusula e anterior ao ENDIF será executada. Se a cláusula ELSE não for definida, todos os comandos entre o IF e o ENDIF serão ignorados. Nesse caso, a execução do programa continua com o primeiro comando seguinte ao ENDIF.
Comandos	Conjunto de comandos ADVPL que serão executados dependendo da avaliação da expressão lógica em lExpressao.



*Fique  
atento*

É possível aninhar um bloco de comando IF...ELSE...ENDIF dentro de outro bloco de comando IF...ELSE...ENDIF. Porém, para a avaliação de mais de uma expressão lógica, deve-se utilizar o comando DO CASE...ENDCASE, ou a versão estendida da expressão IF...ELSE...ENDIF denominada IF...ELSEIF...ELSE...ENDIF.

Exemplo:

```
Local dVencto := CTOD("31/12/01")
If Date() > dVencto
  Alert("Vencimento ultrapassado!")
Endif
Return
```

## O Comando IF...ELSEIF...ELSE...ENDIF

---

Executa o primeiro conjunto de comandos cuja expressão condicional resulta em verdadeiro (.T.).

### Sintaxe

```
IF lExpressao1
```

```
    Comandos
```

```
    [ELSEIF lExpressaoX
```

```
        Comandos]
```

```
    [ELSE
```

```
        Comandos...]
```

```
ENDIF
```

### Parâmetros

lExpressao1	Especifica uma expressão lógica que é avaliada. Se lExpressao resultar em verdadeiro (.T.), executará os comandos compreendidos entre o IF e a próxima expressão da estrutura (ELSEIF ou IF).  Se lExpressao resultar em falso (.F.), será avaliada a próxima expressão lógica vinculada ao comando ELSEIF, ou se o mesmo não existir, será executada a ação definida no comando ELSE.
lExpressaoX	Especifica uma expressão lógica que será avaliada para cada comando ELSEIF. Esta expressão somente será avaliada se a expressão lógica, especificada no comando IF, resultar em falso (.F.).  Se a lExpressaoX avaliada resulte em falso (.F.) será avaliada a próxima expressão lExpressaoX, vinculada ao próximo comando ELSEIF, ou se o mesmo não existir será executada a ação definida para o comando ELSE.
Comandos	Conjunto de comandos ADVPL que serão executados, dependendo da avaliação da expressão lógica em lExpressao.



*Fique  
atento*

O campo IF...ELSE...ELSEIF...ENDIF possui a mesma estruturação de decisão que pode ser obtida com a utilização do comando DO CASE...ENDCASE.

Exemplo:

```
Local dVencto := CTOD("31/12/01")
If Date() > dVencto
    Alert("Vencimento ultrapassado!")
ElseIf Date() == dVencto
    Alert("Vencimento na data!")
Else
    Alert("Vencimento dentro do prazo!")
Endif
Return
```

## O Comando DO CASE...ENDCASE

---

Executa o primeiro conjunto de comandos cuja expressão condicional resulta em verdadeiro (.T.).

### Sintaxe

DO CASE

CASE lExpressao1

    Comandos

[CASE lExpressao2

    Comandos

...

CASE lExpressaoN

    Comandos]

[OTHERWISE

    Comandos]

ENDCASE

## Parâmetros

CASE lExpressao1 Comandos...	<p>Quando a primeira expressão CASE, resultante em verdadeiro (.T.) for encontrada, o conjunto de comandos seguinte é executado. A execução do conjunto de comandos continua até que a próxima cláusula CASE, OTHERWISE ou ENDCASE for encontrada. Ao terminar de executar esse conjunto de comandos, a execução continua com o primeiro comando seguinte ao ENDCASE.</p> <p>Se uma expressão CASE resultar em falso (.F.), o conjunto de comandos seguinte a esta até a próxima cláusula é ignorado.</p> <p>Apenas um conjunto de comandos é executado. Estes são os primeiros comandos cuja expressão CASE é avaliada como verdadeiro (.T.). Após a execução, qualquer outra expressão CASE posterior é ignorada (mesmo que sua avaliação resultasse em verdadeiro).</p>
OTHERWISE Comandos	<p>Se todas as expressões CASE forem avaliadas como falso (.F.), a cláusula OTHERWISE determina se um conjunto adicional de comandos deve ser executado. Se essa cláusula for incluída, os comandos seguintes serão executados e então o programa continuará com o primeiro comando seguinte ao ENDCASE. Se a cláusula OTHERWISE for omitida, a execução continuará normalmente após a cláusula ENDCASE.</p>



*Fique  
atento*

O Comando DO CASE...ENDCASE é utilizado no lugar do comando IF...ENDIF, quando um número maior do que uma expressão deve ser avaliada, substituindo a necessidade de mais de um comando IF...ENDIF aninhados.

Exemplo:

```
Local nMes      := Month(Date())
Local cPeriodo := ""

DO CASE
CASE nMes <= 3
cPeriodo := "Primeiro Trimestre"
CASE nMes >= 4 .And. nMes <= 6
cPeriodo := "Segundo Trimestre"
CASE nMes >= 7 .And. nMes <= 9
cPeriodo := "Terceiro Trimestre"
OTHERWISE
cPeriodo := "Quarto Trimestre"
ENDCASE

Return
```



### Exercícios

#### Exercício 04

Desenvolver um programa que implemente o algoritmo de descascar batatas, utilizando a estrutura de repetição FOR, demonstrando quantas batatas foram descascadas:



### Exercícios

#### Exercício 05

Desenvolver um programa que implemente o algoritmo de descascar batatas, utilizando a estrutura de repetição FOR, demonstrando quantas batatas faltam para serem descascadas:



### Exercícios

#### Exercício 06

Desenvolver um programa que implemente o algoritmo do Jogo da Forca:



### Exercícios

#### Exercício 07

Desenvolver um programa que implemente o algoritmo do Jogo da Velha:



Anotações

---

---

---

---

## **6. ARRAYS E BLOCOS DE CÓDIGO**

### **6.1. Arrays**

Arrays , ou matrizes são coleções de valores, semelhantes a uma lista. Uma matriz pode ser criada através de diferentes maneiras. Cada item em um array é referenciado pela indicação de sua posição numérica na lista, iniciando pelo número 1.

O exemplo a seguir declara uma variável, atribui um array de três elementos a ela, e então exibe um dos elementos e o tamanho do array:

```
Local aLetras          // Declaração da variável  
aLetras := {"A", "B", "C"} // Atribuição do array a variável  
Alert(aLetras[2])      // Exibe o segundo elemento do array  
Alert(cValToChar(Len(aLetras))) // Exibe o tamanho do array
```

O ADVPL permite a manipulação de arrays facilmente. Enquanto que em outras linguagens como C ou Pascal é necessário alocar memória para cada elemento de um array (o que tornaria a utilização de "ponteiros" necessária), o ADVPL se encarrega de gerenciar a memória e torna simples adicionar elementos a um array, utilizando a função AADD():

```
AADD(aLetras,"D") // Adiciona o quarto elemento ao final do array.  
Alert(aLetras[4]) // Exibe o quarto elemento.  
Alert(aLetras[5]) // Erro! Não há um quinto elemento no array.
```

### Arrays como Estruturas

Uma característica interessante do ADVPL é que um array pode conter qualquer tipo de dado: números, datas, lógicos, caracteres, objetos, etc., e ao mesmo tempo. Em outras palavras, os elementos de um array não precisam ser necessariamente do mesmo tipo de dado, em contraste com outras linguagens como C e Pascal.

```
aFunct1 := {"Pedro",32,.T.}
```

Este array contem uma string, um número e um valor lógico. Em outras linguagens como C ou Pascal, este "pacote" de informações pode ser chamado como um "struct" (estrutura em C, por exemplo) ou um "record" (registro em Pascal, por exemplo). Como se fosse na verdade um registro de um banco de dados, um pacote de informações construído com diversos campos. Cada campo tendo uma parte diferente de um dado.

Suponha que no exemplo anterior, o array aFunct1 contenha informações sobre o nome de uma pessoa, sua idade e sua situação matrimonial. Os seguintes #defines podem ser criados para indicar cada posição dos valores dentro de um array:

```
#define FUNCT_NOME 1  
#define FUNCT_IDADE 2  
#define FUNCT_CASADO 3
```

E considere mais alguns arrays para representar mais pessoas:

```
aFunct2 := {"Maria" , 22, .T.}  
aFunct3 := {"Antônio", 42, .F.}
```

Os nomes podem ser impressos assim:

```
Alert(aFunct1[FUNCT_NOME])  
Alert(aFunct2[FUNCT_NOME])  
Alert(aFunct3[FUNCT_NOME])
```

Agora, ao invés de trabalhar com variáveis individuais, é possível agrupá-las em um outro array, do mesmo modo que muitos registros são agrupados em uma tabela de banco de dados:

```
aFuncs := {aFunct1, aFunct2, aFunct3}
```

Que é equivalente a isso:

```
aFuncs := { {"Pedro" , 32, .T.}, ;  
           {"Maria" , 22, .T.}, ;  
           {"Antônio", 42, .F.} }
```

aFuncs é um array com 3 linhas por 3 colunas. Uma vez que as variáveis separadas foram combinadas em um array, os nomes podem ser exibidos assim:

```
Local nCount  
For nCount := 1 To Len(aFuncs)  
    Alert(aFuncs[nCount, FUNCT_NOME])  
    // O acesso a elementos de um array multidimensional.  
    // pode ser realizado também desta forma:  
    // aFuncs[nCount][FUNCT_NOME]  
    Next nCount
```

A variável nCount seleciona que funcionário (ou que linha) é de interesse. Então a constante FUNCT\_NOME seleciona a primeira coluna daquela linha.

## Cuidados com Arrays

---

Arrays são listas de elementos, portanto memória é necessária para armazenar estas informações. Como estes arrays podem ser multidimensionais, a memória necessária será a multiplicação do número de itens em cada dimensão do array, considerando-se o tamanho do conteúdo de cada elemento contido nesta. Portanto o tamanho de um array pode variar muito.

A facilidade da utilização de arrays, mesmo que para armazenar informações em pacotes como descrito anteriormente, não é compensada pela utilização em memória quando o número de itens em um array for muito grande. Quando o número de elementos for muito grande deve-se procurar outras soluções, como a utilização de um arquivo de banco de dados temporário.

### **6.1.1.**

### **Inicializando arrays**

Algumas vezes o tamanho da matriz é conhecido previamente. Outras vezes, o tamanho do array somente será conhecido em tempo de execução.

---

Se o tamanho do array é conhecido

Se o tamanho do array é conhecido no momento que o programa é escrito, há diversas maneiras de implementar o código:

```
01 Local nCnt  
02 Local aX[10]  
03 Local aY := Array(10)  
04 Local aZ := {0,0,0,0,0,0,0,0,0,0}  
05  
06 For nCnt := 1 To 10  
07     aX[nCnt] := nCnt * nCnt  
08 Next nCnt
```

Este código preenche o array com uma tabela de quadrados. Os valores serão 1, 4, 9, 16 ... 81, 100. Note que a linha 07 se refere à variável aX, mas poderia também trabalhar com aY ou aZ.

O objetivo deste exemplo é demonstrar três modos de criar um array de tamanho conhecido, no momento da criação do código.

1. Na linha 02 o array é criado usando aX[10]. Isto indica ao ADVPL para alocar espaço para 10 elementos no array. Os colchetes [e ] são utilizados para indicar o tamanho necessário.
2. Na linha 03 é utilizada a função array com o parâmetro 10 para criar o array, e o retorno desta função é atribuído à variável aY. Na linha 03 é efetuado o que se chama "desenhar a imagen do array". Como se pode notar, existem dez 0's na lista encerrada entre chaves ({ }). Claramente, este método não é o utilizado para criar uma matriz de 1000 elementos.

3. O terceiro método difere dos anteriores porque inicializa a matriz com os valores definitivos. Nos dois primeiros métodos, cada posição da matriz contém um valor nulo (Nil) e deve ser inicializado posteriormente.

4. A linha 07 demonstra como um valor pode ser atribuído para uma posição existente em uma matriz, especificando o índice entre colchetes.

## Se o tamanho do array não é conhecido

Se o tamanho do array não é conhecido até o momento da execução do programa, há algumas maneiras de criar um array e adicionar elementos a ele. O exemplo a seguir ilustra a idéia da criação de um array vazio (sem nenhum elemento), e adição de elementos dinamicamente.

```
01 Local nCnt
02 Local aX[0]
03 Local aY := Array(0)
04 Local aZ := {}
05
06 For nCnt := 1 To nSize
07     AADD(aX, nCnt*nCnt)
08 Next nCnt
```

1. A linha 02 utiliza os colchetes para criar um array vazio. Apesar de não ter nenhum elemento, seu tipo de dado é array.

2. Na linha 03 a chamada da função array cria uma matriz sem nenhum elemento.

3. Na linha 04 está declarada a representação de um array vazio em ADVPL. Mais uma vez, estão sendo utilizadas as chaves para indicar que o tipo de dados da variável é array. Note que {} é um array vazio (tem o tamanho 0), enquanto {Nil} é um array com um único elemento nulo (tem tamanho 1).

Porque cada uma destes arrays não contém elementos, a linha 07 utiliza a função AADD() para adicionar elementos sucessivamente até o tamanho necessário (especificado por exemplo na variável nSize).

## **6.1.2.**

### **Funções de manipulação de arrays**

A linguagem ADVPL possui diversas funções que auxiliam na manipulação de arrays, dentre as quais podemos citar as mais utilizadas:

ARRAY()  
AADD()  
ACLONE()  
ADEL()  
ASIZE()  
AINS()  
ASORT()  
ASCAN()

## ARRAY()

Sintaxe	ARRAY(nLinhas, nColunas)
Descrição	A função Array() é utilizada na definição de variáveis de tipo array, como uma opção a sintaxe, utilizando chaves ("{}").

## AADD()

Sintaxe	AADD(aArray, xItem)
Descrição	A função AADD() permite a inserção de um item em um array já existente, sendo que este item pode ser um elemento simples, um objeto, ou outro array.

## ACLONE()

Sintaxe	AADD(aArray)
Descrição	A função ACLONE() realiza a cópia dos elementos de um array para outro array integralmente.

## ADEL()

Sintaxe	ADEL(aArray, nPosição)
Descrição	A função ADEL() permite a exclusão de um elemento do array. Ao efetuar a exclusão de um elemento, todos os demais são reorganizados de forma que a ultima posição do array passará a ser nula.

## ASIZE()

Sintaxe	ASIZE(aArray, nTamanho)
Descrição	A função ASIZE permite a redefinição da estrutura de um array pré-existente,

	adicionando ou removendo itens do mesmo.
--	--

## ASORT()

Sintaxe	ASORT(aArray, nInicio, nItens, bOrdem)
Descrição	A função ASORT() permite que os itens de um array sejam ordenados, a partir de um critério pré-estabelecido.

## ASCAN()

Sintaxe	ASCAN(aArray, bSeek)
Descrição	A função ASCAN() permite que seja identificada a posição do array que contém uma determinada informação, através da análise de uma expressão descrita em um bloco de código.

## AINS()

Sintaxe	AINS(aArray, nPosicao)
Descrição	A função AINS() permite a inserção de um elemento, no array especificado, em qualquer ponto da estrutura do mesmo, diferindo desta forma da função AADD(), a qual sempre insere um novo elemento ao final da estrutura já existente.

### 6.1.3.

### Cópia de arrays

Conforme comentado anteriormente, um array é uma área na memória, o qual possui uma estrutura que permite as informações serem armazenadas e organizadas das mais diversas formas.

Para “copiar” o conteúdo de uma variável, utiliza-se o operador de atribuição “:=”, conforme abaixo:

```
nPessoas := 10  
nAlunos := nPessoas
```

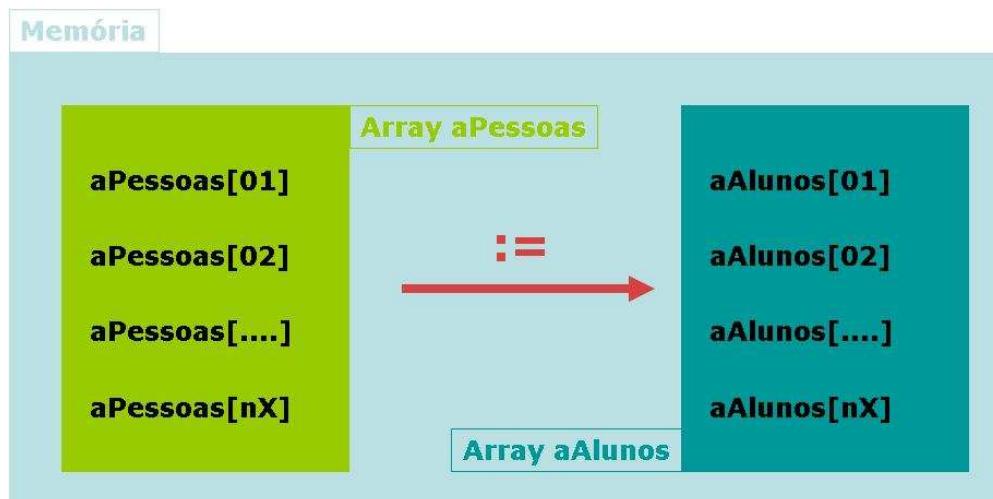
Ao executar a atribuição de nAlunos com o conteúdo de nPessoas, o conteúdo de nPessoas é atribuído a variável nAlunos, causando o efeito de cópia do conteúdo de uma variável para outra.

Isto porque o comando de atribuição copia o conteúdo da área de memória, representada pelo nome “nPessoas” para a área de memória representada pelo nome “nAlunos”. Mas ao utilizar o operador de atribuição “:=”, da mesma forma que utilizado em variáveis simples, para se copiar um array o efeito é diferente:

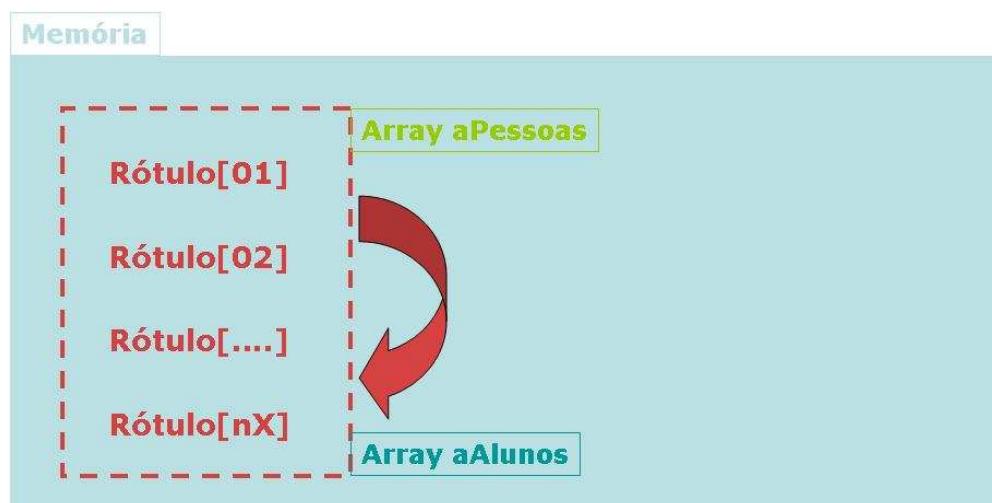
```
aPessoas := {"Ricardo", "Cristiane", "André", "Camila"}  
aAlunos := aPessoas
```

A variável aPessoas representa uma área de memória que contém a estrutura de um array (“mapa”), não as informações do array, pois cada informação está em sua própria área de memória.

Com base nesse conceito, o array pode ser considerado apenas como um “mapa” ou um “guia” de como as informações estão organizadas e de como elas podem ser armazenadas ou consultadas. Para se copiar um array deve-se levar este conceito em consideração, pois caso contrário o resultado esperado não será obtido na execução da “cópia”.

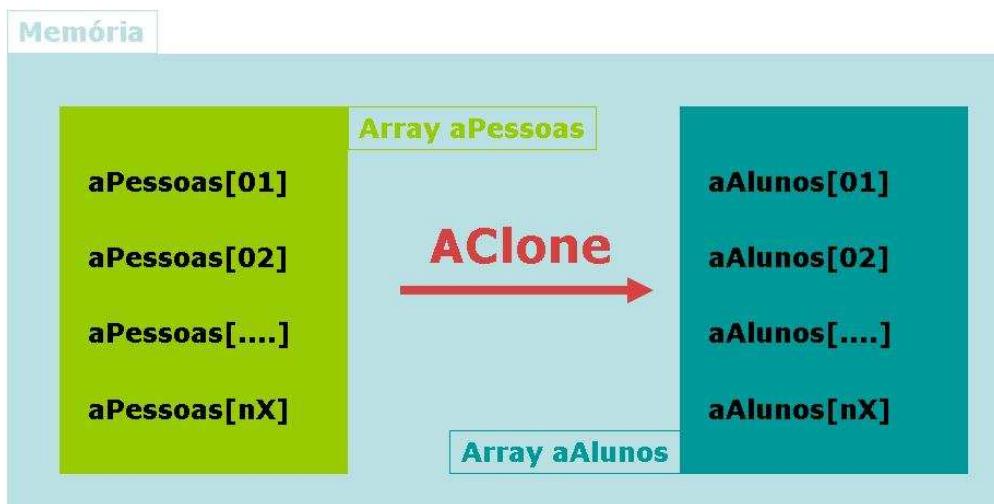


Desta forma, ao atribuir o conteúdo representado pela variável **aPessoas**, a variável **aAlunos** não está se “copiando” as informações e sim o “mapa” das áreas de memória, em que as informações estão realmente armazenadas.



Como foi copiado o “mapa” e não as informações, qualquer ação utilizando o rótulo aAlunos afetará as informações do rótulo aPessoas. Com isso ao invés de se obter dois arrays distintos, tem-se o mesmo array com duas formas de acesso (rótulos) diferentes.

Por esta razão deve ser utilizado o comando AClone(), quando se deseja obter um array com a mesma estrutura e informações que compõe outro array já existente





## **6.2. Listas de Expressões e Blocos de Código**

Bloco de código é um conceito existente há muito tempo em linguagens xBase. Não como algo que apareceu da noite para o dia, e sim uma evolução progressiva utilizando a combinação de muitos conceitos da linguagem para a sua implementação.

Primeira premissa

O ADVPL é uma linguagem baseada em funções. Funções têm um valor de retorno. Assim como o operador de atribuição :=.

Assim, ao invés de escrever:

```
x := 10 // Atribui o valor 10 à variável chamada X  
Alert("Valor de x: " + cValToChar(x))
```

Escreve-se:

```
// Atribui e então exibe o valor da variável X  
Alert("Valor de x: " + cValtoChar(X := 10))
```

A expressão x:=10 é avaliada primeiro, e então seu resultado (o valor de X, que agora é 10) é passada para a função cvaltochar para a conversão para caractere, e em seguida para a função alert para a exibição. Por causa dessa regra de precedência é possível atribuir um valor a mais de uma variável ao mesmo tempo:

```
Z := Y := X := 0
```

Por causa dessa regra, essa expressão é avaliada como se fosse escrita assim:

```
Z := ( Y := (X := 0) )
```

Apesar de o ADVPL avaliar expressões da esquerda para a direita, no caso de atribuições, acontece ao contrário, ou seja, da direita para a esquerda. O valor é atribuído à variável X, que retorna o valor para ser atribuído à variável Y e assim sucessivamente. O zero foi "propagado através da expressão".

### Segunda premissa

Em ADVPL junta-se diversas linhas de código em uma única linha física de comando. Por exemplo, o código:

```
If IACHOU  
  Alert("Cliente encontrado!")  
Endif  
pode ser escrito assim:  
If IACHOU ; Alert("Cliente encontrado!");  
Endif
```

O ponto-e-vírgula indica ao ADVPL que a nova linha de código está para começar. É possível então colocar diversas linhas lógicas de código na mesma linha física, através do editor de texto utilizado.

Apesar da possibilidade de se escrever todo o programa assim, em uma única linha física, isto não é recomendado, pois dificulta a legibilidade do programa e, consequentemente, a sua manutenção.

## 6.2.2.

### Lista de expressões

A evolução dos blocos de código começa com as listas de expressões. Nos exemplos a seguir, o símbolo ==> indicará o retorno da expressão após a sua avaliação (seja para atribuir em uma variável, exibir para o usuário ou imprimir em um relatório), que será impressa em um relatório, por exemplo.

Duas Linhas de Código

```
@00,00 PSAY x := 10    ==>    10  
@00,00 PSAY y := 20    ==>    20
```

Cada uma das linhas terá a expressão avaliada, e o valor da variável será então impresso.

Duas linhas de código em uma , utilizando -se ponto-e-vírgula

Este é o mesmo código que o anterior, apenas escrito em uma única linha:

```
Alert( cValToChar( x := 10 ; y := 20 ) )    ==>    10
```

Apesar desse código se encontrar em uma única linha física, existem duas linhas lógicas separadas pelo ponto e vírgula. Ou seja, esse código é equivalente a:

```
Alert( cValToChar( x := 10 ) )  
y := 20
```

Portanto, apenas o valor 10 da variável x será passado para as funções cvaltochar e alert para ser exibido. E o valor 20 apenas será atribuído à variável y.

## Convertendo para uma lista de expressões

Quando parênteses são colocados ao redor do código e o sinal de ponto-e-vírgula substituído por uma vírgula apenas, o código torna-se uma lista de expressões:

```
Alert( cValToChar( ( X := 10 , Y := 20 ) ) ) ==> 20
```

O valor de retorno resultante de uma lista de expressões é o valor resultante da última expressão ou elemento da lista. Funciona como se fosse um pequeno programa ou função, que retorna o resultado de sua última avaliação (efetuadas da esquerda para a direita).

Nesse exemplo, a expressão `x := 10` é avaliada, e então a expressão `y := 20`, cujo valor resultante é passado para a função `alert` e `cvaltochar`, e então exibido. Depois que essa linha de código é executada, o valor de `X` é igual a 10 e o de `y` igual a 20, e 20 será exibido.

Teoricamente, não há limitação para o número de expressões que podem ser combinadas em uma lista. Na prática, o número máximo é por volta de 500 símbolos.

Debugar listas de expressões é difícil porque as expressões não estão divididas em linhas de código fonte, o que torna todas as expressões associadas a uma mesma linha de código. Isto pode tornar muito difícil determinar onde um erro ocorreu.

Onde utilizar uma lista de expressões?

~~O propósito principal de uma lista de expressões é agrupá-las em uma única unidade. Em qualquer lugar do código ADVPL, em que uma expressão simples pode ser utilizada, pode-se utilizar uma lista de expressões. E ainda, fazer com que várias coisas aconteçam onde normalmente apenas uma aconteceria.~~

```
X := 10 ; Y := 20
```

```
If X > Y  
  Alert("X")  
Else  
  Alert("Y")  
Z := -1  
Endif
```

Aqui temos o mesmo conceito, escrito utilizando listas de expressões na função IIF():

```
X := 10 ; Y := 20
iif( X > Y, ;
( Alert("X"), Z := 1 ), ;
( Alert("Y"), Z := -1 ) )
```

De listas de expressões para blocos de código

---

Considere a seguinte lista de expressões:

```
Alert( cValToChar( ( x := 10, y := 20 ) ) ) ==> 20
```

O ADVPL permite criar funções, que são pequenas partes de um código, como se fosse um pequeno programa, utilizados para diminuir em partes de tarefas mais complexas e reaproveitar código em mais de um lugar num programa. Para maiores detalhes consulte a documentação sobre a criação de funções em ADVPL. Porém, a idéia neste momento é que a lista de expressões, utilizada na linha anterior, pode ser criada como uma função:

```
Function Lista()
X := 10
Y := 20
Return Y
```

E a linha de exemplo, com a lista de expressões, pode ser substituída, tendo o mesmo resultado, por:

```
Alert( cValToChar( Lista() ) ) ==> 20
```

Como mencionado anteriormente, uma lista de expressões é como um pequeno programa ou função. Com poucas mudanças, uma lista de expressões pode se tornar um bloco de código:

```
( X := 10 , Y := 20 ) // Lista de Expressões
{|| X := 10 , Y := 20 } // Bloco de Código
```

Note as chaves {} utilizadas no bloco de código. Ou seja, um bloco de código é uma matriz.  
Porém na verdade, não é uma lista de dados, e sim uma lista de comandos, uma lista de código.

```
// Isto é uma matriz de dados
A := {10, 20, 30}
// Isto é um bloco de código, porém funciona como // se fosse uma matriz de comandos
B := {|| x := 10, y := 20}
```

### 6.2.3.

### Blocos de Código

Diferentemente de uma matriz, não se pode acessar elementos de um bloco de código, através de um índice numérico. Porém blocos de código são semelhantes a uma lista de expressões, e a uma pequena função.

Ou seja, podem ser executados. Para a execução, ou avaliação de um bloco de código, deve-se utilizar a função Eval():

```
nRes := Eval(B) ==> 20
```

Essa função recebe como parâmetro um bloco de código e avalia todas as expressões contidas neste bloco de código, retornando o resultado da última expressão avaliada.

#### Passando Parâmetros

Já que os blocos de código são como pequenas funções, também é possível a passagem de parâmetros para um bloco de código. Os parâmetros devem ser informados entre as barras verticais (||), separados por vírgulas, assim como em uma função.

```
B := { | N | X := 10, Y := 20 + N }
```

Porém deve-se notar que já que o bloco de código recebe um parâmetro, um valor deve ser passado quando o bloco de código for avaliado.

```
C := Eval(B, 1) ==> 21
```

#### Utilizando Blocos de Código

Blocos de código podem ser utilizados em diversas situações. Geralmente são utilizados para executar tarefas quando os eventos de objetos são acionados ou, para modificar o comportamento padrão de algumas funções. Como no exemplo, considere a matriz abaixo:

```
A := {"GARY HALL", "FRED SMITH", "TIM JONES"}
```

Esta matriz pode ser ordenada pelo primeiro nome, utilizando-se a chamada da função asort(A), resultado na matriz com os elementos ordenados dessa forma:

```
{"FRED SMITH", "GARY HALL", "TIM JONES"}
```

A ordem padrão para a função asort é ascendente. Este comportamento pode ser modificado através da informação de um bloco de código que ordena a matriz de forma descendente:

```
B := { |X, Y| X > Y }  
aSort(A, B)
```

O bloco de código (de acordo com a documentação da função asort) deve ser escrito para aceitar dois parâmetros que são os dois elementos da matriz para comparação. Note que o bloco de código não conhece que elementos estão comparando - a função asort seleciona os elementos (talvez utilizando o algoritmo QuickSort) e passa-os para o bloco de código. O bloco de código compara-os e retorna verdadeiro (.T.) se encontram na ordem correta, ou falso (.F.) se não. Se o valor de retorno for falso, a função asort então trocará os valores de lugar e seguirá comparando o próximo par de valores.

Então, no bloco de código anterior, a comparação  $X > Y$  é verdadeira se os elementos estão em ordem descendente, o que significa que o primeiro valor é maior que o segundo.

Para ordenar a mesma matriz pelo último nome, também em ordem descendente, é possível utilizar o seguinte bloco de código:

```
B := { |X, Y| SUBSTR(X, At(" ",X)+1) > SUBSTR(Y, At(" ",Y)+1) }
```

Note que este bloco de código procura e compara as partes dos caracteres, imediatamente seguinte a um espaço em branco. Depois de utilizar esse bloco de código para a função asort, a matriz conterá:

```
{"GARY HALL", "TIM JONES", "FRED SMITH"}
```

Finalmente, para ordenar um sub-elemento (coluna) de uma matriz por exemplo, utiliza-se o seguinte bloco de código:

```
B := { |X, Y| X[1] > Y[1] }
```

## **6.2.4.**

### **Funções para manipulação de blocos de código**

A linguagem ADVPL possui diversas funções que auxiliam na manipulação de blocos de código, dentre as quais podemos citar as mais utilizadas:

EVAL()  
DBEVAL()  
AEVAL()  
EVAL()

---

Sintaxe	EVAL(bBloco, xParam1, xParam2, xParamZ)
Descrição	A função EVAL() é utilizada para a avaliação direta de um bloco de código, utilizando as informações disponíveis no momento de sua execução. Esta função permite a definição e passagem de diversos parâmetros que serão considerados na interpretação do bloco de código.

### **DBEVAL()**

---

Sintaxe	Array(bBloco, bFor, bWhile)
Descrição	A função DBEval() permite que todos os registros, de uma determinada tabela, sejam analisados e para cada registro será executado o bloco de código definido.

### **AEVAL()**

---

Sintaxe	AEVAL(aArray, bBloco, nInicio, nFim)
Descrição	A função AEVAL() permite que todos os elementos de um determinada array sejam analisados e para cada elemento será executado o bloco de código definido.



Anotações

---

---

---

---

## 7. FUNCÕES

A maior parte das rotinas escritas em programas são compostas de um conjunto de comandos. Essas rotinas se repetem ao longo de todo o desenvolvimento. Uma função nada mais é do que um conjunto de comandos que para ser utilizada, basta chamá-la pelo seu nome.

Para tornar uma função mais flexível, ao chamá-la é possível a passagem de parâmetros, que contêm os dados e as informações que definem o processamento da função.

Os parâmetros das funções descritas utilizando a linguagem ADVPL são posicionais, ou seja, na sua passagem não importa o nome da variável e sim a sua posição dentro da lista de parâmetros, que permite executar uma função escrevendo:

```
Calcula(parA, parB, parC) // Chamada da função em uma rotina
```

E a função deve estar escrita assim:

```
User Function Calcula(x, y, z)
```

```
... Comandos da Função
```

```
Return ...
```

Neste caso, x assume o valor de parA, y de parB e z de parC.

A função também tem a faculdade de retornar uma variável, podendo inclusive ser um Array.

Para tal, encerra-se a função com:

```
Return(campo)
```

```
Assim A := Calcula(parA,parB,parC) atribui a A o conteúdo do retorno da função Calcula.
```

No ADVPL existem milhares de funções escritas pela equipe de Tecnologia Microsiga, pelos analistas de suporte e pelos próprios usuários.



*Fique  
atento*

Existe um ditado que diz:

“Vale mais um programador que conhece todas as funções disponíveis em uma linguagem do que aquele que, mesmo sendo gênio, reinventa a roda a cada novo programa”.

No TDN (TOTVS Developer Network) mais de 500 estão documentadas, e este número tende a aumentar exponencialmente com os novos processos de documentação que estão em implantação na Tecnologia e Inteligência Protheus.

O objetivo do curso é apresentar, demonstrar e fixar a utilização das principais funções, sintaxes e estruturas utilizadas em ADVPL.

No ADVPL até os programas chamados pelo menu são funções, sendo que em um repositório não pode haver funções com o mesmo nome e para permitir que os usuários e analistas possam desenvolver suas próprias funções sem que as mesmas conflitem com as já disponíveis no ambiente ERP, foi implementada pela Tecnologia Microsiga um tipo especial de função denominado “User Function”.

Nos tópicos a seguir serão detalhados os tipos de funções disponíveis na linguagem ADVPL, suas formas de utilização e respectivas diferenças.

## **7.1. Tipos e escopos de funções**

Em ADVPL podem ser utilizados os seguintes tipos de funções:

Function()  
User Function()  
Static Function()  
Main Function()

Function()

Funções ADVPL convencionais, restritas ao desenvolvimento da área de Inteligência Protheus da Microsiga.

O interpretador ADVPL distingue os nomes de funções do tipo Function() com até dez caracteres. A partir do décimo caracter, apesar do compilador não indicar quaisquer tipos de erros, o interpretador ignorará os demais caracteres.

**Exemplo:**

```
// Fonte MATA100INCL.PRW  
#INCLUDE "protheus.ch"
```

```
Function MATA100INCL01()
```

```
    ALERT("01")
```

```
    Return
```

```
Function MATA100INCL02()
```

```
    ALERT("02")
```

```
    Return
```

Ao executar a função MATA100INCL01() será exibida a mensagem “01”, mas ao executar a função MATA100INCL02() também será exibida a mensagem “01”, pois o interpretador considera o nome da função como “MATA100INC”.



*Fique  
atento*

Funções do tipo Function() somente podem ser executadas através dos módulos do ERP.

Somente poderão ser compiladas funções do tipo Function() se o MP-IDE possuir uma autorização especial fornecida pela Microsiga.

Funções do tipo Function() são acessíveis por quaisquer outras funções em uso pela aplicação.

## User Function()

---

As “User Defined Functions” ou funções definidas pelos usuários, são tipos especiais de funções implementados pelo ADVPL para garantir que desenvolvimentos específicos não realizados pela Inteligência Protheus da Microsiga sobreponham as funções padrões desenvolvidas para o ERP.

O interpretador ADVPL considera que o nome de uma User Function é composto pelo nome definido para a função, precedido dos caracteres “U\_”. Desta forma a User Function XMAT100I será tratada pelo interpretador como “U\_XMAT100I”.



*Fique  
atento*

Como ocorre o acréscimo dos caracteres “U\_” no nome da função e o interpretador considera apenas os dez primeiros caracteres da função, para sua diferenciação é recomendado que os nomes das User Functions tenham apenas oito caracteres, evitando resultados indesejados durante a execução da aplicação.



*Dica*

Funções do tipo User Function são acessíveis por quaisquer outras funções em uso pela aplicação, desde que em sua chamada sejam utilizados os caracteres “U\_”, em conjunto com o nome da função.

As User Functions podem ser executadas a partir da tela inicial do client do ERP (Microsiga Protheus Remote), mas as aplicações que pretendem disponibilizar esta opção devem possuir um preparo adicional de ambiente.

Para maiores informações consulte no TDN o tópico sobre preparação de ambiente e a documentação sobre a função RpcSetEnv().

## Static Function()

Funções ADVPL tradicionais, cuja visibilidade está restrita às funções descritas no mesmo arquivo de código fonte no qual estão definidas.

Exemplo:

```
//Fonte FINA010.PRW

Function FINA010()

CriaSx1("FIN010")
Return

Static Function CRIASX1()
//Fonte FINA020.PRW
Function FINA020()

CriaSx1("FIN020")
Return

Static Function CRIASX1()
```

No exemplo acima, existem duas funções denominadas CRIASX1(), definidas em arquivos de código fonte distintos: FINA010.PRW e FINA020.PRW.

A função FINA010() terá visibilidade apenas da função CRIASX1(), definida no arquivo de código fonte FINA010.PRW, sendo que o mesmo ocorre com a função FINA020().

Este recurso permite isolar funções de uso exclusivo de um arquivo de código fonte, evitando a sobreposição ou a duplicação de funções na aplicação.

Neste contexto as Static Functions() são utilizadas para:

1. Padronizar o nome de uma determinada função, que possui a mesma finalidade, mas que sua implementação pode variar de acordo com a necessidade da função principal / aplicação.
2. Redefinir uma função padrão da aplicação, adequando-a às necessidades específicas de uma função principal / aplicação.
3. Proteger as funções de uso específico de um arquivo de código fonte / função principal.



O Ambiente de desenvolvimento utilizado na aplicação ERP (MP-IDE) valida se existem Functions(), Main Functions() ou User Functions() com o mesmo nome, mas em arquivos de códigos fontes distintos, evitando a duplicidade ou sobreposição de funções.

### Main Function()

---

Main Function() é outro tipo de função especial do ADVPL incorporado para permitir tratamentos diferenciados na aplicação ERP.

Uma Main Function() tem a característica de poder ser executada através da tela inicial de parâmetros do client do ERP (Microsiga Protheus Remote), da mesma forma que uma User Function, com a diferença que as Main Functions somente podem ser desenvolvidas com o uso da autorização de compilação, tornando sua utilização restrita à Inteligência Protheus da Totvs.

Na aplicação ERP é comum o uso das Main Functions(), nas seguintes situações:

1. Definição dos módulos da aplicação ERP: Main Function Sigaadv()
2. Definição de atualizações e updates: AP710TOMP811()
3. Atualizações específicas de módulos da aplicação ERP: UpdateATF()

## **7.2. Passagem de parâmetros entre funções**

Como mencionado anteriormente os parâmetros das funções descritas, utilizando a linguagem ADVPL são posicionais, ou seja, na sua passagem não importa o nome da variável e sim a sua posição dentro da lista de parâmetros.

Complementando esta definição, podem ser utilizadas duas formas distintas de passagens de parâmetros para funções descritas na linguagem ADVPL:

Passagem de parâmetros por conteúdo.

Passagem de parâmetros por referência.

Passagem de parâmetros por conteúdo

A passagem de parâmetros por conteúdo é a forma convencional de definição dos parâmetros recebidos pela função chamada, na qual a função recebe os conteúdos passados pela função chamadora, na ordem com os quais são informados.

```
User Function CalcFator(nFator)
```

```
Local nCnt
```

```
Local nResultado := 0
```

```
For nCnt := nFator To 1 Step -1
```

```
nResultado *= nCnt
```

```
Next nCnt
```

```
Alert("O fatorial de " + cValToChar(nFator) + ;  
" é " + cValToChar(nResultado))
```

```
Return
```

Avaliando a função CalcFator() descrita anteriormente, podemos verificar que a mesma recebe como parâmetro para a sua execução à variável nFator.

Com base nesta função, podemos descrever duas formas de passagem de parâmetros por conteúdo:

Passagem de conteúdos diretos.

Passagem de variáveis como conteúdos.

#### Exemplo 01 – Passagem de conteúdos diretos

```
User Function DirFator()
```

```
Local nResultado := 0
```

```
nResultado := CalcFator(5)
```

A passagem de conteúdos diretos implica na definição explícita do valor do parâmetro, na execução da chamada da função. Nesse caso foi informado o conteúdo 5 (numérico) como conteúdo para o primeiro parâmetro da função CalcFator.

Como a linguagem ADVPL trata os parâmetros de forma posicional, o conteúdo 5 será atribuído diretamente à variável, definida como primeiro parâmetro da função chamada. No nosso caso, nFator.

Por ser uma atribuição de parâmetros por conteúdo, o interpretador da linguagem basicamente executa uma operação de atribuição normal, ou seja, nFator := 5.



*Fique  
atento*

Duas características da linguagem ADVPL tornam necessária uma atenção especial na chamada de funções:

A linguagem ADVPL não é uma linguagem “tipada”, de forma que as variáveis não tem um tipo previamente definido, aceitando o conteúdo que lhes for imposto por meio de uma atribuição.

Os parâmetros de uma função são atribuídos de acordo com a ordem em que tais parâmetros são definidos na chamada desta ordem. Não é realizado nenhum tipo de consistência em relação aos tipos dos conteúdos, e a obrigatoriedade de parâmetros nesta ação.



*Dica*

Os parâmetros de uma função são caracterizados como variáveis de escopo LOCAL para efeito de execução.

Dessa forma os mesmos não devem ser definidos novamente como LOCAL, na área de definição e inicialização de variáveis, pois se isso ocorrer haverá a perda dos valores recebidos pela redefinição das variáveis na função.

Se for necessário garantir um conteúdo padrão para um determinado parâmetro, deve ser utilizado o identificador DEFAULT, conforme detalhamento no tópico “Tratamento de valores padrões para parâmetros de funções”.

**Exemplo 02 – Passagem de variáveis como conteúdos**

User Function DirFator()

Local nResultado := 0

Local nFatorUser := 0

nFatorUser := GetFator() // Função ilustrativa na qual o usuário informa o fator a ser utilizado.

nResultado := CalcFator(nFatorUser)

A passagem de conteúdos como variáveis implica na utilização de variáveis de apoio para executar a chamada de uma função. Nesse caso foi informada a variável nFatorUser, a qual será definida pelo usuário com a utilização da função ilustrativa GetFator(). O uso de variáveis de apoio flexibiliza a chamada de outras funções, pois elas serão parametrizadas de acordo com as necessidades daquele processamento específico, no qual se encontra a função chamadora.

Como a linguagem ADVPL trata os parâmetros de forma posicional, o conteúdo da variável nFatorUser será atribuído diretamente à variável definida como primeiro parâmetro da função chamada, no caso nFator.

Por ser uma atribuição de parâmetros por conteúdo, o interpretador da linguagem basicamente executa uma operação de atribuição normal, ou seja, nFator := nFatorUser.



Dica

A passagem de parâmetros não necessita que as variáveis informadas na função chamadora tenham os mesmos nomes das variáveis utilizadas na definição de parâmetros da função chamada.

Dessa forma podemos ter:

```
User Function DirFator()
```

```
Local nFatorUser := GetFator()
```

```
nResultado := CalcFator(nFatorUser)
```

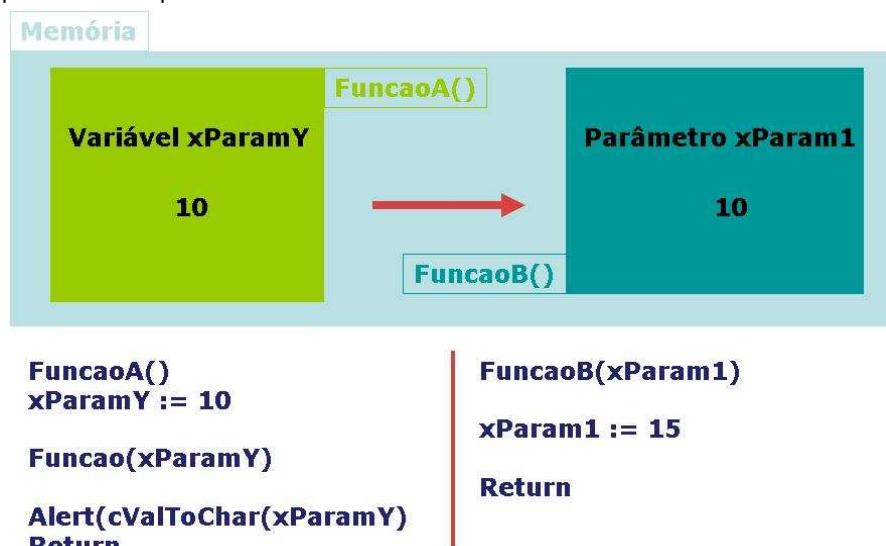
```
Function CalcFator(nFator)
```

As variáveis nFatorUser e nFator podem ter nomes diferentes pois o interpretador fará a atribuição de conteúdo com base na ordem dos parâmetros e não pelo nome das variáveis.

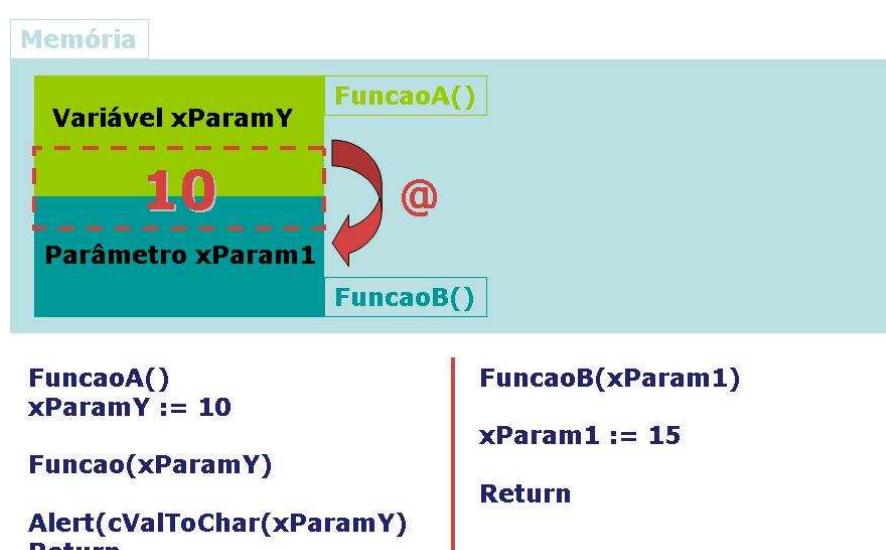
## Passagem de parâmetros por referência

A passagem de parâmetros por referência é uma técnica muito comum nas linguagens de programação, pois permite que variáveis de escopo LOCAL tenham o seu conteúdo manipulado por funções específicas, mantendo o controle dessas variáveis restritas à função que as definiu e às funções desejadas pela aplicação.

A passagem de parâmetros por referência utiliza o conceito de que uma variável é uma área de memória e, portanto, passar um parâmetro por referência nada mais é do que, ao invés de passar o conteúdo para a função chamada, passar qual a área de memória utilizada pela variável passada.



Passagem de parâmetros tradicional – Duas variáveis x Duas áreas de memória



Passagem de parâmetros por referência – Duas variáveis x uma única área de memória



Anotações

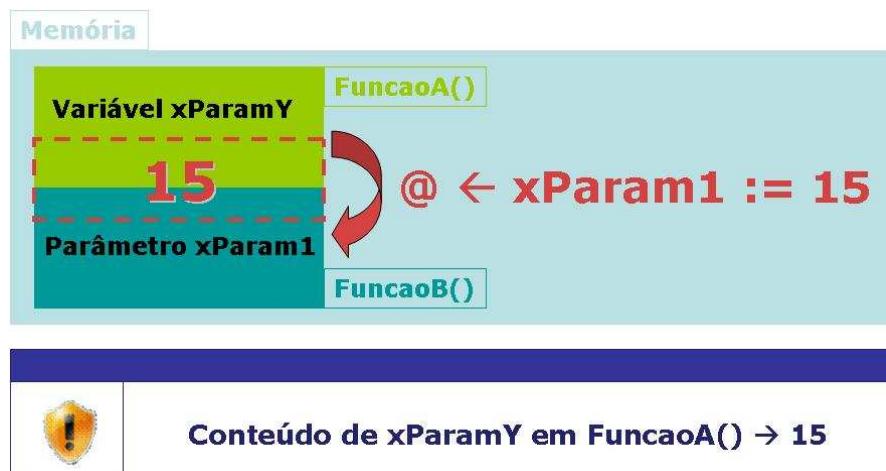
---

---

---

---

Dessa forma, a função chamada tem acesso não apenas ao conteúdo, mas à variável em si, pois a área de memória é a variável e qualquer alteração será visível à função chamadora quando tiver o retorno desta função.



### Tratamento de conteúdos padrões para os parâmetros de funções

O tratamento de conteúdos padrão para parâmetros de funções é muito utilizado nas funções padrões da aplicação ERP, de forma a garantir a correta execução dessas funções por qualquer função chamadora, evitando situações de ocorrências de erros pela falta da definição de parâmetros necessários à correta utilização da função.



*Dica*

A linguagem ADVPL não obriga a passagem de todos os parâmetros descritos na definição da função, sendo que os parâmetros não informados serão considerados com conteúdo nulo.

Dessa forma, o uso do identificador DEFAULT permite ao desenvolvedor a garantia de que, na utilização da função, determinados parâmetros terão o valor com um tipo adequado à função.

#### Exemplo:

```
User Function CalcFator(nFator)
```

```
Local nCnt
```

```
Local nResultado := 0
```

```
Default nFator := 1
```

```
For nCnt := nFator To 1 Step -1
```

```
    nResultado *= nCnt
```

```
Next nCnt
```

```
Return nResultado
```

No exemplo descrito, se o parâmetro nFator não for informado na função chamadora, o mesmo terá seu conteúdo definido como 1.

Se esse tratamento não fosse realizado e com isso o parâmetro nFator não fosse informado, ocorreria o seguinte evento de erro:

**Exemplo:**

```
User Function CalcFator(nFator)
```

```
Local nCnt
```

```
Local nResultado := 0
```

```
For nCnt := nFator To 1 Step -1 // nFator está como Nulo, portanto nCnt é nulo
```

```
    nResultado *= nCnt
```

```
Next nCnt // Ao efetuar o Next, o interpretador realiza a ação nCnt += 1.
```

```
Return nResultado
```

Como o interpretador realizará a ação `nCnt += 1`, e o conteúdo da variável `nCnt` é nulo, ocorrerá o erro de "type mismatch on `+=`, expected N      U", pois os tipos das variáveis envolvidos na operação são diferentes: `nCnt`      nulo (U) e `1`      numérico (N).



*Dica*

Se o parâmetro que possui a opção `DEFAULT`, descrita no fonte, for informado, a linha de `DEFAULT` não será executada, mantendo-se dessa forma o conteúdo passado pela função chamadora.



*Anotações*

---

---

---

---

## **8. DIRETIVAS DE COMPILAÇÃO**

O compilador ADVPL possui uma funcionalidade denominada pré-processador. O pré-processador é um programa que examina o programa fonte escrito em ADVPL e executa certas modificações nele, baseadas nas Diretivas de Compilação.

As diretivas de compilação são comandos que não são compilados, sendo dirigidos ao pré-processador, o qual é executado pelo compilador antes da execução do processo de compilação propriamente dito. Portanto, o pré-processador modifica o programa fonte, entregando para o compilador um programa modificado de acordo com as diretivas de compilação. Estas são iniciadas pelo caractere “#”.

As diretivas podem ser colocadas em qualquer parte do programa, sendo que as implementadas pela linguagem ADVPL são:

```
#INCLUDE  
#DEFINE  
#IFDEF  
#IFNDEF  
#ELSE  
#ENDIF  
#COMMAND
```



Lem  
bre-  
se

As diretivas de compilação também são conhecidas como UDC – User Defined Commands.

### Diretiva: #INCLUDE

A diretiva #INCLUDE indica em que arquivo de extensão “CH” (padrão ADVPL) estão os UDCs a serem utilizados pelo pré-processador.

A aplicação ERP possui diversos includes que devem ser utilizados segundo a aplicação que será desenvolvida. Dessa forma, será permitida a utilização de recursos adicionais definidos para a linguagem, implementados pela área de Tecnologia da Totvs.

Os includes mais utilizados nas aplicações ADVPL, desenvolvidas para o ERP são:

PROTHEUS.CH: Diretivas de compilação como padrões para a linguagem. Contém a especificação da maioria das sintaxes utilizadas nos fontes, inclusive permitindo a compatibilidade da sintaxe tradicional do Clipper para os novos recursos implementados no ADVPL.

O include PROTHEUS.CH ainda contém a referência a outros includes utilizados pela linguagem ADVPL que complementam essa funcionalidade de compatibilidade com a sintaxe Clipper, tais como:

- DIALOG.CH
- FONT.CH
- INI.CH
- PTMENU.CH
- PRINT.CH



A utilização do include “protheus.ch”, nos fontes desenvolvidos para a aplicação ERP Protheus, é obrigatória e necessária ao correto funcionamento das aplicações

AP5MAIL.CH: Permite a utilização da sintaxe tradicional na definição das seguintes funções de envio e recebimento de e-mail:

- CONNECT SMTP SERVER
- CONNECT POP SERVER
- DISCONNECT SMTP SERVER
- DISCONNECT POP SERVER
- POP MESSAGE COUNT
- SEND MAIL FROM
- GET MAIL ERROR
- RECEIVE MAIL MESSAGE

TOPCONN.CH: Permite a utilização da sintaxe tradicional na definição das seguintes funções de integração com a ferramenta TOPCONNECT (MP10 – DbAccess):

- TCQUERY

TBICONN.CH: Permite a utilização da sintaxe tradicional na definição de conexões, com a aplicação Server do ambiente ERP, através das seguintes sintaxes:

- CREATE RPCCCONN
- CLOSE RPCCCONN
- PREPARE ENVIRONMENT
- RESET ENVIRONMENT
- OPEN REMOTE TRANSACTION
- CLOSE REMOTE TRANSACTION
- CALLPROC IN
- OPEN REMOTE TABLES

XMLXFUN.CH: Permite a utilização da sintaxe tradicional, na manipulação de arquivos e strings no padrão XML, através das seguintes sintaxes:

- CREATE XMLSTRING
- CREATE XMLFILE
- SAVE XMLSTRING
- SAVE XMLFILE
- ADDITEM TAG
- ADDNODE NODE
- DELETENODE



Dica

Os recursos de tratamentos de e-mails, integração com a ferramenta TOPCONNECT (DbAccess), preparação de ambientes e manipulação de arquivos e strings do padrão XML, serão abordados no curso de ADVPL Avançado.



**Dica**

O diretório de includes deve ser especificado no Ambiente de desenvolvimento do ERP Protheus (MP-IDE), para cada configuração de compilação disponível.

Se o diretório de includes não estiver informado, ou esteja informado incorretamente, será exibida uma mensagem de erro informando:

“Não foi possível criar o arquivo <caminho\nome> .ERX”



**Dica**

As funções desenvolvidas para a aplicação ERP costumam utilizar includes, para definir o conteúdo de strings e as variáveis diversas, utilizadas pela aplicação em diferentes idiomas. Dessa forma é normal verificar que um fonte possui um arquivo “.CH” com o mesmo nome, o que caracteriza esse tipo de include.

## **Diretiva: #DEFINE**

A diretiva #DEFINE permite que o desenvolvedor crie novos termos para serem utilizados no código fonte. Este termo tem o efeito de uma variável de escopo PUBLIC, mas que afeta somente o fonte na qual o #DEFINE está definido com a característica de não permitir a alteração de seu conteúdo.

Dessa forma, um termo definido através da diretiva #DEFINE pode ser considerado como uma constante.



**Dica**

Os arquivos de include definidos para os fontes da aplicação ERP contêm diretivas #DEFINE para as strings de textos de mensagens exibidas para os usuários nos três idiomas com os quais a aplicação é distribuída:

Português, Inglês e Espanhol.

Por essa razão a aplicação ERP possui três repositórios distintos para cada uma das bases de dados homologadas pela Microsiga, pois cada compilação utiliza uma diretiva referente ao seu idioma.

## **Diretivas: #IFDEF, IFNDEF, #ELSE e #ENDIF**

As diretivas #IFDEF, #IFNDEF, #ELSE e #ENDIF permitem ao desenvolvedor criar fontes flexíveis e sensíveis a determinadas configurações da aplicação ERP.

Com essas diretivas, podem ser verificados parâmetros do Sistema, tais como o idioma com o qual está parametrizado e a base de dados utilizada para armazenar e gerenciar as informações do ERP. Assim, ao invés de escrever dois, ou mais códigos fontes que realizam a mesma função, mas utilizando recursos distintos para cada base de dados, ou exibindo mensagem para cada um dos idiomas tratados pela aplicação. O desenvolvedor pode preparar seu código fonte para ser avaliado pelo pré-processador, que irá gerar um código compilado de acordo com a análise dos parâmetros de ambiente.

Essas diretivas de compilação estão normalmente associadas às seguintes verificações de ambiente:

Idioma: verifica as variáveis SPANISH e ENGLISH, disponibilizadas pela aplicação. O idioma português é determinado pela exceção:

```
#IFDEF SPANISH
    #DEFINE STR0001 "Hola !!!"
#ELSE

    #IFDEF ENGLISH
        #DEFINE STR0001 "Hello !!!"
    #ELSE
        #DEFINE STR0001 "Olá !!!"
    #ENDIF

#ENDIF
```



Fique atento

Apesar da estrutura semelhante ao IF-ELSE-ELSEIF-ENDIF, não existe a diretiva de compilação #ELSEIF, o que torna necessário o uso de diversos #IFDEFs para a montagem de uma estrutura que seria facilmente solucionada com IF-ELSE-ELSEIF-ENDIF.



Dica

A aplicação ERP disponibiliza para a variável de escopo PUBLIC - \_\_LANGUAGE, que contém uma string que identifica o idioma em uso pelo Sistema, cujos conteúdos possíveis são:

“PORTUGUESE”

“SPANISH”

“ENGLISH”

Banco de Dados: Verifica as variáveis AXS e TOP para determinar se o banco de dados em uso pela aplicação está no formato ISAM (DBF, ADS, CTREE, etc.) ou, se está utilizando a ferramenta TOPCONNECT (DbAcess).

```
#IFDEF TOP

    cQuery := "SELECT * FROM "+RETSQLNAME("SA1")
    dbUseArea(.T., "TOPCONN", TcGenQry(,,cQuery), "SA1QRY", .T., .T.)

#else
    DbSelectArea("SA1")
#endif
```



Fique  
atento

Os bancos de dados padrão AS400 não permitem a execução de queries no formato SQLANSI, através da ferramenta TOPOCONNECT (DbAcess).

Dessa forma é necessário realizar uma verificação adicional ao #IFDEF TOP antes de executar uma query, que no caso é realizada através do uso da função TcSrvType(), a qual retorna a string “AS/400”, quando este for o banco em uso.

Para esses bancos deve ser utilizada a sintaxe ADVPL tradicional.

## Diretiva: #COMMAND

A diretiva #COMMAND é utilizada principalmente nos includes da linguagem ADVPL para efetuar a tradução de comandos em sintaxe CLIPPER, para as funções implementadas pela Tecnologia Microsiga.

Essa diretiva permite que o desenvolvedor defina para o compilador como uma expressão deve ser interpretada.

Trecho do arquivo PROTHEUS.CH

```
#xcommand @ <nRow>, <nCol> SAY [ <oSay> <label: PROMPT, VAR> ] <cText> ;  
[ PICTURE <cPict> ] ; [ <dlg: OF, WINDOW, DIALOG> <oWnd> ] ;  
[ FONT <oFont> ] ; [ <lCenter: CENTERED, CENTER> ] ;  
[ <lRight: RIGHT> ] ; [ <lBorder: BORDER> ] ;  
[ <lPixel: PIXEL, PIXELS> ] ; [ <color: COLOR, COLORS> <nClrText> [, <nClrBack> ] ] ;  
[ SIZE <nWidth>, <nHeight> ] ; [ <design: DESIGN> ] ;  
[ <update: UPDATE> ] ; [ <lShaded: SHADED, SHADOW> ] ;  
[ <lBox: BOX> ] ; [ <lRaised: RAISED> ] ;  
=> ;
```

```
[ <oSay> := ] TSay():New( <nRow>, <nCol>, <{cText}>,;  
[<oWnd>], [<cPict>], <oFont>, <.lCenter.>, <.lRight.>, <.lBorder.>,;  
<.lPixel.>, <nClrText>, <nClrBack>, <nWidth>, <nHeight>;  
<.design.>, <.update.>, <.lShaded.>, <.lBox.>, <.lRaised.> )
```

Através da diretiva #COMMAND, o desenvolvedor determinou as regras para que a sintaxe tradicional da linguagem CLIPPER, para o comando SAY, fosse convertida na especificação de um objeto TSAY() do ADVPL.



### Exercícios

#### Exercício 08

Vamos Desenvolver um programa que permita ao usuário pesquisar um cliente, informando seu CNPJ e se o mesmo existir na base, exibir suas principais informações.



### Exercícios

#### Exercício 09

Utilizando a interface visual desenvolvida para o exercício anterior, desenvolver a função genérica GetTexto(), para ser utilizada nas aplicações do Jogo da Velha e Jogo da Forca.



### Exercícios

#### Exercício 10

Utilizando a interface visual desenvolvida para o exercício anterior, desenvolver a função genérica GetTexto(), para ser utilizada nas aplicações do Jogo da Velha e Jogo da Forca.



### Exercícios

#### Exercício 11

Utilizando a função AVISO(), desenvolver um programa que permita ao usuário selecionar a opção de busca de CNPJ por cliente ou fornecedor, e se encontrar, exiba os seus dados principais.



## Exercícios

### Exercício 12

Desenvolver uma rotina que capture vários CNPJs de clientes informados pelo usuário, e verifique para cada um deles se o mesmo existe ou não na base de dados. Ao final informar quais CNPJs foram informados, e de acordo com a seleção do usuário, exibir os dados principais de um destes clientes.



### Exercícios

#### Exercício 13

Utilizando a função FORMBATCH(), desenvolver uma rotina que verifique se para cada item de uma nota fiscal de entrada existe o respectivo cabeçalho, e se for encontrado algum item inconsistente, comunique a ocorrência ao usuário que está realizando o processamento.



### Exercícios

#### Exercício 14

Vamos Desenvolver uma rotina que, através do uso de um bloco de código, converta a estrutura da tabela SA1, obtida com a função DBSTRUCT(), em uma string denominada cCampo.

## MÓDULO 03: DESENVOLVENDO PEQUENAS CUSTOMIZAÇÕES

### **9. ADVPL E O ERP MICROSIGA PROTHEUS**

O ADVPL (Advanced Protheus Language) é uma linguagem de programação desenvolvida pela Totvs e que contém todas as instruções e funções necessárias ao desenvolvimento de um sistema, independentemente de sua complexidade.

O Sistema Protheus, por outro lado, é uma plataforma tecnológica que engloba um Servidor de Aplicação, um Dicionário de Dados e as Interfaces para a conexão com o usuário. É o Protheus que executa o código ADVPL, assim como o devido acesso à base de dados.

O Protheus é composto pelo ERP (que engloba, além das funcionalidades descritas nos capítulos anteriores, mais de trinta verticais aplicadas às áreas específicas de negócios) e pelo Configurador (programa que permite customizar o Sistema às necessidades do usuário de forma fácil).

## 9.1. O Ambiente Protheus

O Sistema Protheus é constituído de um conjunto de Softwares que compõem as camadas de funcionalidades básicas aos serviços de aplicação, interface, banco de dados e repositório, conforme o diagrama da figura abaixo:

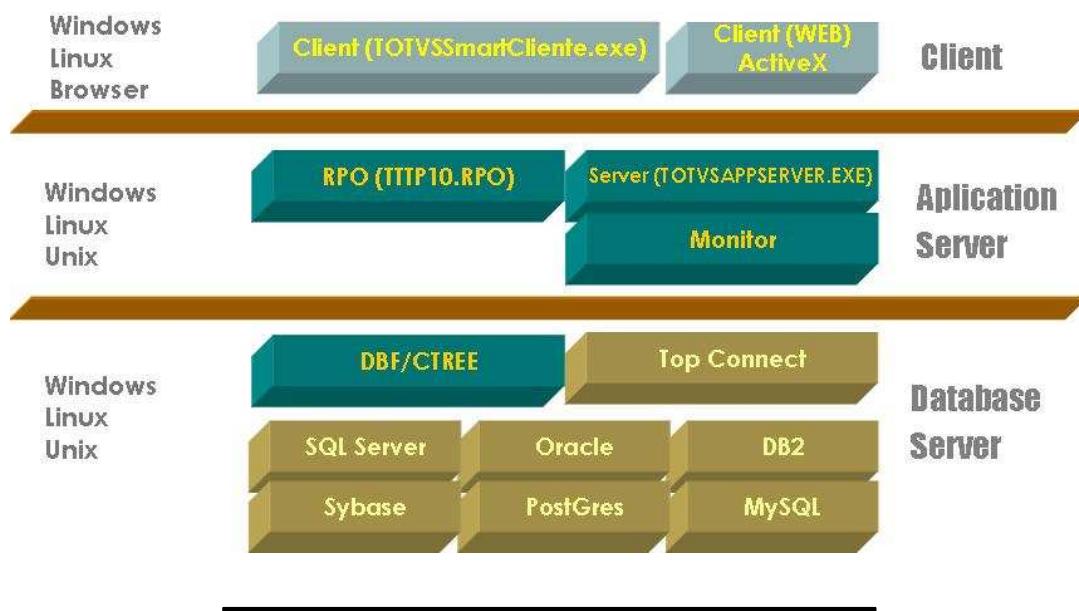


Figura: Camadas básicas do Ambiente Protheus

Para executar um programa desenvolvido em ADVPL, é preciso antes de mais nada escrevê-lo e compilá-lo. Este procedimento é feito através da ferramenta TOTVS DevStudio do Protheus (Totvs Development Studio).

O objetivo do TOTVS DevStudio é facilitar a tarefa de escrever programas: através de cores, indica se a palavra escrita é uma instrução, uma variável, ou um comentário; organiza a biblioteca de programas em projetos e administra o repositório de objetos; aponta erros de sintaxe; permite o debug (execução passo a passo do programa, verificando o conteúdo das variáveis) e fornece assistentes (modelos) de programas.

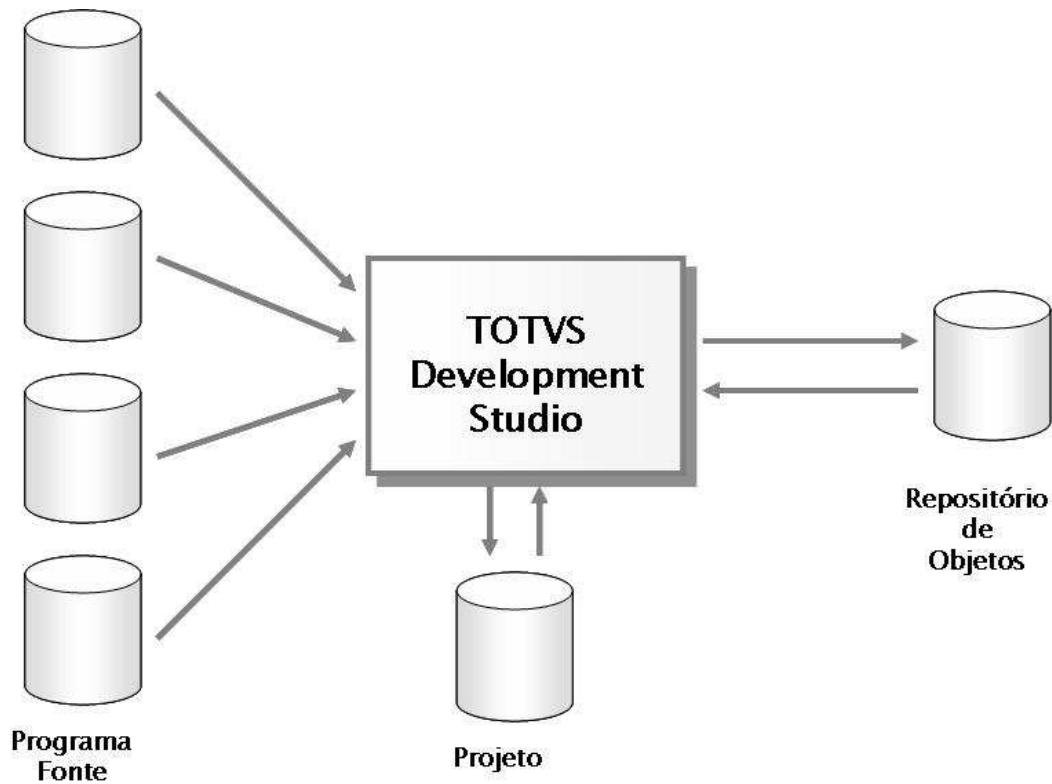


Figura: Manutenção no repositório de objetos

Após compilar o programa, o resultado é um objeto que é carregado na memória ficando disponível para sua execução através da aplicação PROTHEUS.

O objeto não é um executável, ou seja, não está convertido para a linguagem nativa do equipamento. Quem faz esse trabalho é o Protheus Server em tempo de execução. Por isso, o Protheus Server está sempre presente na memória em tempo de execução, permitindo:

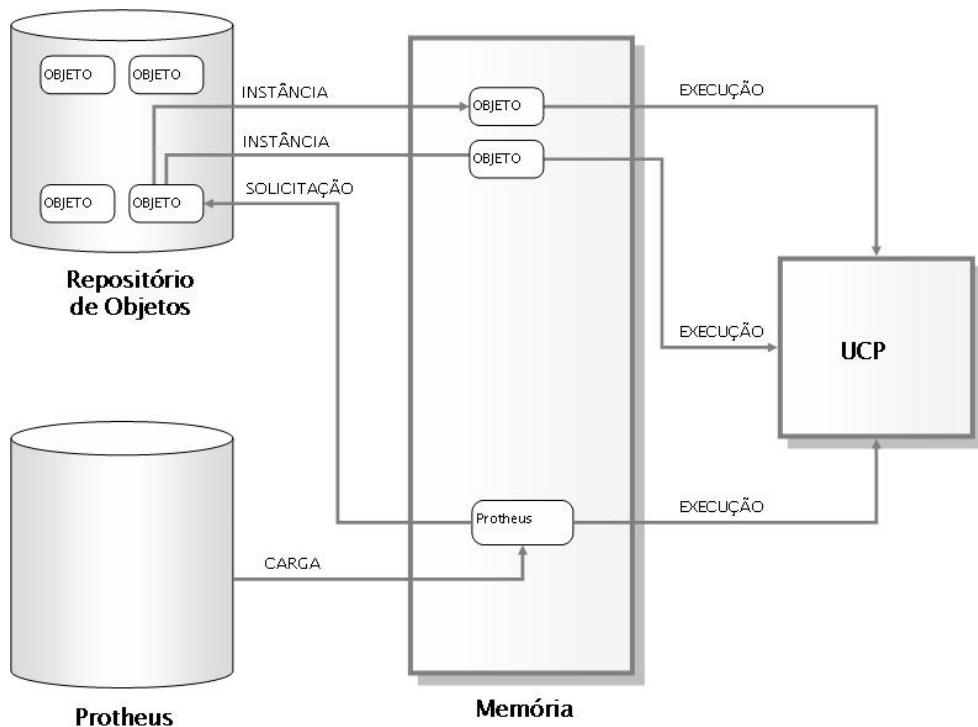
Proteger o programa fonte, evitando que seja alterado indevidamente, pois somente os objetos são distribuídos com uma execução mais rápida em função da compilação no DEV-Studio;

Flexibilização à plataforma de trabalho. Assim, um mesmo programa pode rodar em Ambientes Windows, Linux,

ou mesmo em um Hand Held, ficando a tarefa de adequação para o Servidor Protheus;

Que o Sistema cresça de forma ilimitada, pois os objetos ficam fora do executável;

O uso de macro substituições, ou seja, o uso de rotinas exteriores ao Sistema, armazenadas em arquivos e que podem facilmente ser alteradas pelo usuário, pois o Server também interpreta o código fonte em tempo de execução.



---

Figura: Diagrama esquemático de objetos Protheus

O Repositório de Objetos é a biblioteca de objetos de todo o Ambiente Protheus, incluindo tanto os objetos implementados para as funcionalidades básicas do ERP como aqueles gerados pelos usuários. A figura abaixo demonstra a estrutura e a interconexão entre as várias camadas.

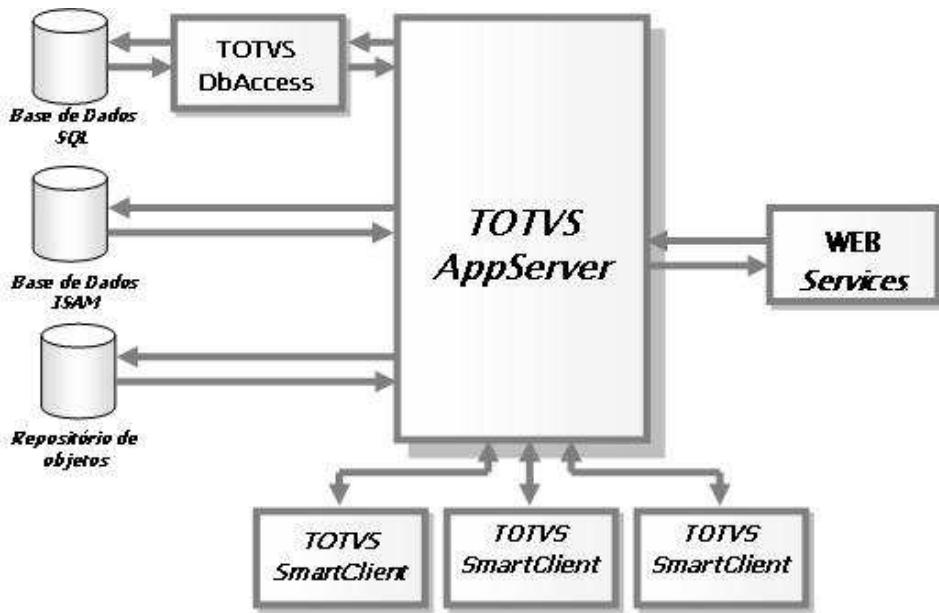


Figura: Estrutura de interconexão do Protheus

Ela demonstra também que os dados a serem processados podem estar armazenados em bases ISAM ou em Bancos de Dados padrão SQL. No primeiro caso, o server comunica-se diretamente com os dados. Em Bancos SQL é a interface TOPCONNECT / DBACCESS que converte os comandos de entrada e saída, adequando-os ao SQL utilizado (SQL Server Microsoft, Oracle, DB2, etc.).

Uma vez terminado o processamento do objeto chamado, ele é descartado da memória, ou seja, o Protheus é um Sistema que pode crescer de forma ilimitada pois os objetos, armazenados em um repositório, praticamente não ocupam espaço no HD (Hard Disk).

O Protheus é uma plataforma multicamada. Entre as diversas camadas, temos a interface de apresentação ao usuário (Remote), o tratamento dado para as regras de negócio implementadas (Server), o acesso aos objetos do repositório (Server), o acesso aos dados disponíveis no Banco de Dados (Server ou TOPCONNECT / DBACCESS) e ao gerenciamento de serviços WEB (Server). Nesse processo, o Protheus possui, basicamente, quatro aplicativos utilizados com diferentes finalidades:

Protheus Server / TOTVS AppServer: Responsável pela comunicação entre o cliente, o banco de dados e o RPO. O nome do executável depende da versão do Sistema (TOTVSAAPPSERVER.EXE) sendo que as plataformas ISAM suportadas pelo Protheus Server são DBF e CTREE.

Protheus Remote / TOTVS SmartClient: Instalado no Server ou na estação. O nome também depende da versão do Sistema (TOTVSSMARTCLIENT.EXE).

TopConnect / DbAccess: Responsável pela conversão dos comandos de banco de dados, adequando-os ao SQL utilizado.

Protheus Monitor / TOTVS Monitor: Programa de análise que verifica quem está usando o Sistema e possibilita o envio de mensagens, ou mesmo derrubar conexões (TOTVSMONITOR.EXE).

Alguns nomes referem-se a um conjunto de programas para facilitar a sua identificação:

RPO: É o arquivo binário do APO (Advanced Protheus Objects), ou seja, os objetos.

Build: Executáveis, DLLs e o RPO completo.

Patch: Atualizações pontuais do RPO, aplicadas por meio do IDE.

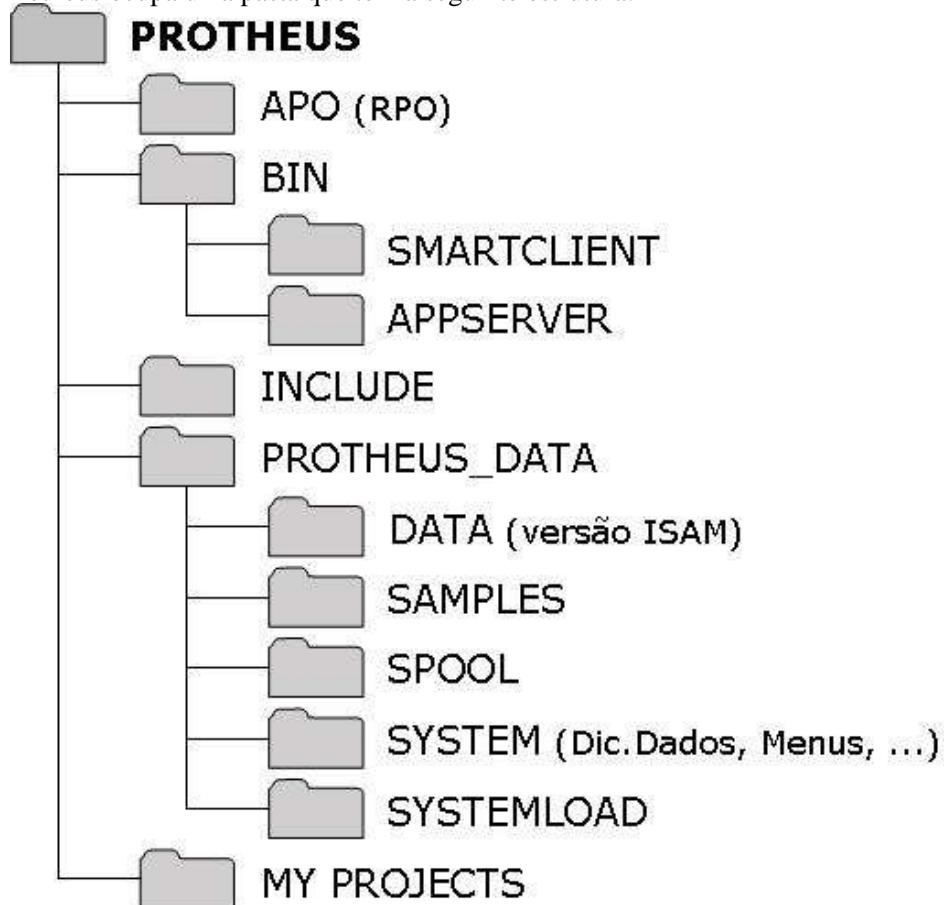
Update: Pacote de atualização para o repositório (RPO), liberado periodicamente contendo todas as adequações e melhorias disponibilizadas para o Sistema em um determinado período, sendo não cumulativo, aplicadas por meio do DEV-Studio.

A interface de apresentação é realizada pelo SmartClient que processa a parte da estação, basicamente, tela e teclado. Pode estar gravado no Server e ser carregado via rede para a memória da estação. Ou, de preferência, deve ficar armazenado no HD da estação. Pode também ser carregado pelo Internet Explorer, rodando dentro do próprio browser com o SmartClient ActiveX e permitindo o acesso ao Protheus Server pela Internet, com as mesmas funcionalidades do SmartClient, sendo que o browser precisa suportar o uso da tecnologia ActiveX.

Caso exista algum Firewall ou Proxy entre o WEB Server e o Browser que vai acessar o SmartClient ActiveX, eles deverão ser configurados para permitir o seu download.

## **9.2. Organização e configuração inicial do ambiente Protheus**

O Protheus ocupa uma pasta que tem a seguinte estrutura:



### Figura: Estrutura básica das pastas do Protheus

APO: Contém o arquivo RPO, repositório de objetos do Protheus.

SMARTCLIENT: Reúne um conjunto de arquivos executáveis, dll's e arquivos de configuração do Sistema, para possibilitar o acesso ao servidor.

APP SERVER: Reúne um conjunto de executáveis, dll's e arquivos de configuração do Sistema que compõem o servidor.

INCLUDE: Contém as bibliotecas necessárias para a compilação de programas Protheus.

DATA: Contém a base de dados no caso de versão ISAM.

SAMPLES: Oferece um conjunto de programas exemplo e arquivos ADVPL padrões da Microsiga.

SPOOL: Nesta pasta são gravados os relatórios gerados em disco pelo Sistema Protheus.

SYSTEM: Contém os arquivos de menus, os arquivos de configurações e os arquivos de customizações (SXs) do sistema Protheus.

SYSTEMLOAD: Contém o dicionário de dados em formato TXT. É neste arquivo que estão todos os padrões e formatos para a geração dos arquivos de configurações e de customizações (SXs), conforme a localização de país definida pelo usuário, na entrada do Sistema.

MY PROJECTS: Sugere-se a criação desta pasta para armazenar projetos e fontes das customizações realizadas pelo usuário.

UPDATES: Sugere-se esta pasta para o armazenamento das atualizações a serem aplicadas no Sistema Protheus.

Apesar da estrutura ilustrada anteriormente, indicar que as pastas estão subordinadas à pasta PROTHEUS, é possível que algumas delas possam estar em máquinas diferentes, ou até mesmo em Ambientes computacionais diferentes.

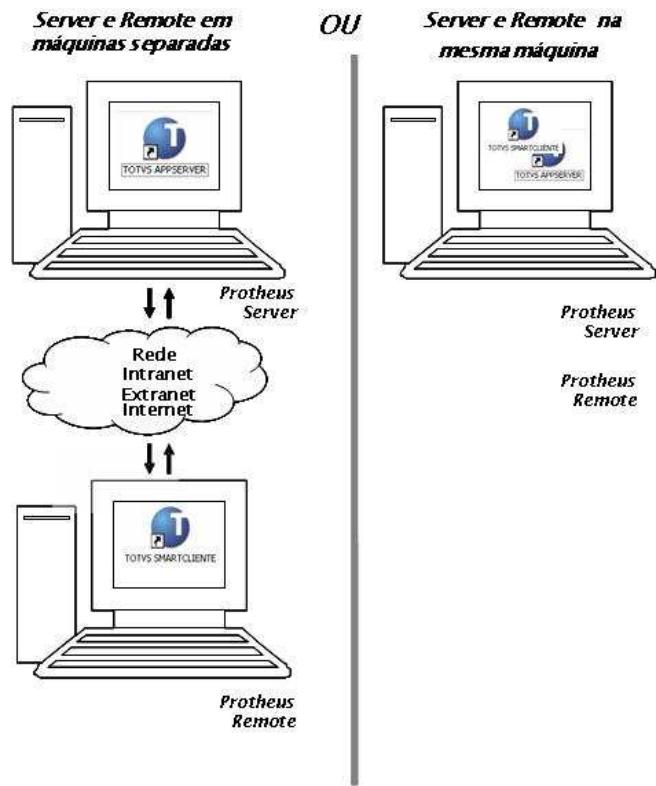


Figura: Formas de instalação e uso do Protheus

Para isso, é necessário configurar, ou seja, informar ao Protheus onde está cada uma delas. Esse tipo de informação consta nos arquivos de parâmetros de configuração do Sistema (TOTVSAPP SERVER.INI e TOTVSSMARTCLIENT.INI) existentes nas respectivas pastas APP SERVER e SMARTCLIENT.

Os parâmetros do TOTVSAPP SERVER.INI são lidos pelo programa TOTVSAPP SERVER.EXE, logo no início de sua execução. O mesmo procedimento ocorre em relação aos parâmetros do TOTVSSMARTCLIENT.INI pelo programa TOTVSSMARTCLIENT.EXE. A execução desses dois programas é feita por meio de ação do usuário, facilitada pelos atalhos TOTVS APP SERVER e TOTVS SMARTCLIENT.

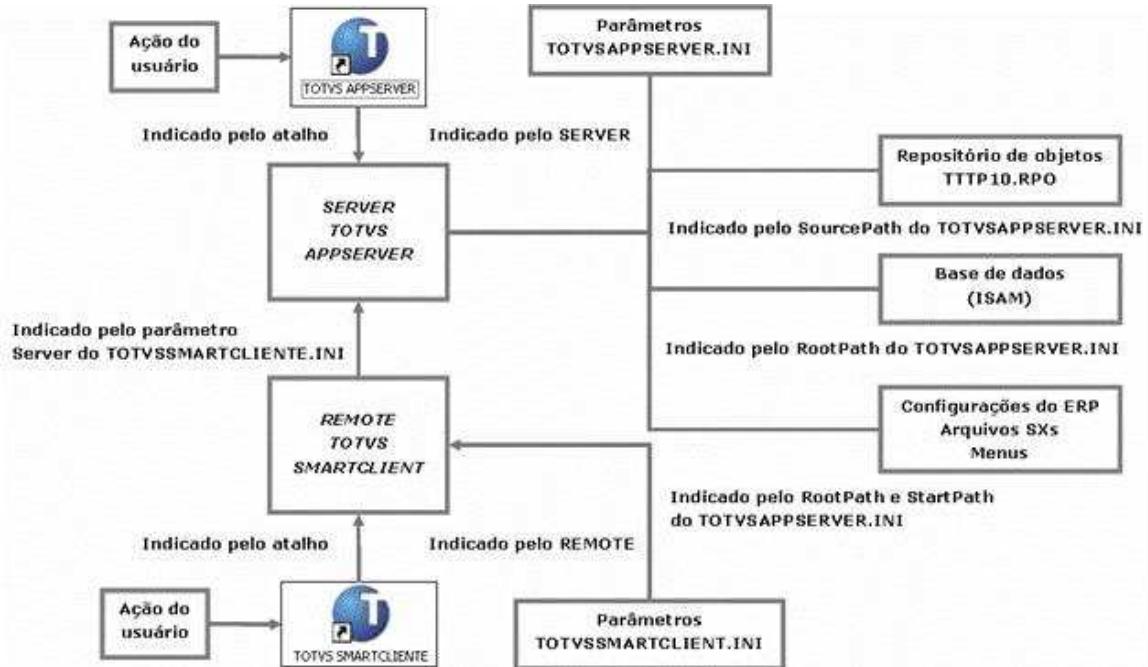


Figura: Links dos parâmetros de configuração

Para que o TOTVS AppServer e o TOTVS SmartClient sejam executados, os arquivos TOTVSAPPERVER.INI e TOTVSMARTCLIENT.INI devem estar disponíveis nas respectivas pastas APPSERVER e SMARTCLIENT, pois são eles que indicam o endereço das demais pastas, conforme a ilustração da figura anterior.

O detalhe de preenchimento das propriedades dos respectivos atalhos TOTVS AppServer e o TOTVS SmartClient é demonstrado a seguir. No atalho do TOTV SAppServer, é necessário que seja informado o parâmetro “-debug” ou “-console”.



## Propriedades dos atalhos



TOTVS APPSERVER

Destino: c:\protheus\bin\appserver\totvsappserver.exe  
- console

Iniciar c:\protheus\bin\appserver  
em:

-Console ou -Debug

Executado como uma JanelaConsole, as informações recebidas das conexões com o TOTVS Application Server conectados são exibidas diretamente na tela do console do TOTVS Application Server, bem como informações de Não Conformidades.

-Install

Se o TOTVS Application Server, não for instalado como um Serviço do NT, durante a Instalação, isto pode ser realizado, executando-o com a opção de Linha de Comando.

-Remove

Para removê-lo da Lista de Serviços do NT, é possível executá-lo com a opção de Linha de Comando.



TOTVS SMARTCLIENTE

Destino: c:\protheus\bin\smartclient\totvssmartcliente.exe  
-M

Iniciar c:\protheus\bin\smartclient  
em:

-Q (Quiet)

Indica que o TOTVS Smart Client não deverá mostrar o Splash (Imagen de Apresentação) e a tela de identificação de Parâmetros Iniciais, necessita ser acompanhada da (Cláusula -P).

**-P (Main Program)**

Identifica o Programa (APO) Inicial.

**-E (Environment)**

Nome da Seção de Environment, no (Ini do Server) que será utilizada, para definições gerais.

**-C (Connection)**

Nome da Seção de Conexão que será utilizada para a conexão ao TOTVS Application Server.

**-M (AllowMultiSession)**

Permite as múltiplas instâncias (Cópias) do TOTVS Smart Client, na mesma máquina, o que por Default não é permitido.

Os parâmetros que configuram o local do RPO, o Banco de Dados (ISAM ou SQL), os arquivos de menus, configurações e customizações do sistema no arquivo INI são:

SourcePath: Indica o local de origem dos objetos. É o endereço do Repositório de Objetos (Exemplo: SourcePath=C:\PROTHEUS\APO).

RootPath: Aponta para a pasta raiz (inicial), a partir da qual serão localizados os dados (no caso de ISAM), bem como o próprio Dicionário de Dados (Exemplo: RootPath=C:\PROTHEUS\PROTHEUS\_DATA).

StartPath: Indica qual é a pasta dentro da pasta raiz (informada no parâmetro RootPath) que contém os arquivos de menus, os arquivos de configurações e os arquivos de customizações (SXs) do Sistema Protheus (Exemplo: StartPath=\SYSTEM\).

Não há necessidade de que os parâmetros estejam em ordem nos arquivos de configuração (.ini). Além dos parâmetros já detalhados, existem outros que podem indicar a versão do Sistema, o tipo de banco de dados, a linguagem do país em que está sendo utilizado e as máscaras de edição e formatação.



```
[ENVIRONMENT]
SOURCEPATHC:\PROTHEU
S\APO
ROOTPATH=
C:\MP811\PROTHEUS_DAT
A
STARTPATH=\PROTHEUS\
RPODB=TOP
RPOLANGUAGE=PORTUGUESE
RPOVERSION=101
LOCALFILES=ADS
TRACE=0
LOCALDBEXTENSION=.DBF
PICTFORMAT=DEFAULT
DATEFORMAT=DEFAULT
```

```
[DRIVERS]
ACTIVE=TCP
```

```
[TCP]
TYPE=TCPIP
PORT=1234
```

Figura: Exemplo de um ambiente em um arquivo de parâmetros

No exemplo da figura anterior, o rótulo [environment] descreve um conjunto de parâmetros que serão inicializados no Sistema. Os rótulos [Drivers] e [TCP] identificam a comunicação que pode ser estabelecida entre o Protheus Server e o Protheus Remote. Outros ambientes podem ser configurados no mesmo arquivo (TOTVSSAPP SERVER.INI).

Já o arquivo de parâmetros do Protheus Remote (TOTVSSSMARTCLIENT.INI) contém apenas as

configurações locais, basicamente as informações necessárias para a inicialização e a comunicação com o Protheus Server, conforme o exemplo da figura a seguir.



[CONFIG]  
LASTMAINPROG=SIGA  
CFG  
LANGUAGE=1

[DRIVERS]  
ACTIVE=TCP

[TCP]  
SERVER=172.16.72.41  
PORT=1234

Figura: Exemplo de um arquivo de configuração do remote

Active: Indica qual é a forma de comunicação.

Port: Indica o número da porta a ser utilizada para a comunicação entre o Protheus Server e o Protheus Remote. É necessário que a porta utilizada na comunicação seja a mesma em ambos (no TOTVSSAPP SERVER.INI e no TOTVSSMARTCLIENT.INI). Vale ressaltar que a porta 80 é reservada para a Internet e pode causar conflitos caso seja utilizada na comunicação do Protheus.

Server: Aponta para o endereço do servidor que pode ser a própria máquina (localhost) ou o nome da máquina (Server= Servidor\_01) ou mesmo um endereço IP (exemplo Server=172.16.72.41).

Exemplo:

O parâmetro Server=172.16.72.41 no arquivo TOTVSSMARTCLIENT.INI indica ao Protheus Remote o endereço da máquina na qual está funcionando o Protheus Server.

## **9.3. O Configurador do Protheus**

### **Funcionalidades Abordadas**

#### **9.3.1.**

O objetivo deste tópico não é abranger toda a estrutura e recursos do módulo Configurador da aplicação ERP, mas permitir a realização de tarefas de configuração simples que serão necessárias no desenvolvimento de pequenas customizações.

Com foco neste objetivo, serão detalhadas as seguintes operações:

Configuração e criação de novas tabelas no Dicionário de Dados.

Atualização das estruturas do Dicionário de Dados.

- tabelas do sistema;
- validações de campos;
- índices de tabelas;
- gatilhos de campos.

Para contextualizar a estrutura da aplicação ERP no tópico a seguir é detalhada a forma como as tabelas de dados do Sistema estão divididas entre os diversos módulos que compõe o Sistema Protheus.

#### **9.3.2. Estruturas básicas da aplicação ERP Protheus**

Arquivos de configuração do sistema

Arquivo	Descrição
SIGAMAT	Cadastro de empresas e filiais do sistema
SIGAPSS	Arquivo de usuários, grupos e senhas do sistema
SIX	Índices dos arquivos
SX1	Perguntas e respostas
SX2	Mapeamento de tabelas
SX3	Dicionário de Dados

SX4	Agenda do Schedule de processos
SX5	Tabelas
SX6	Parâmetros
SX7	Gatilhos de Interface
SX8	Fora de uso
SX9	Relacionamentos entre tabelas
SXA	Pastas cadastrais apontadas no SX3
SXB	Consulta por meio da tecla F3 (Consulta Padrão)
SXD	Controle do Schedule de processos
SXE	Sequência de documentos (+1)
SXF	Sequência de documentos (Próximo)
SXG	Tamanho padrão para campos apontado pelo SX3
SXK	Resposta de Perguntas (SX1) por usuários
SXO	Controle de LOGs por tabela
SXP	Histórico de Logs cadastrados no SXO
SXQ	Cadastro de filtros inteligentes da mbrowse (contém as informações necessárias para a criação do filtro).
SXR	Cadastro de relacionamento entre programa x filtro (utilizada internamente pelo Protheus para verificar em quais programas os filtros poderão ser utilizados).
SXS	Cadastro de programas (utilizado na validação para mostrar/inibir os filtros na execução da mbrowse).
SXT	Tabela de usuários (contém as informações dos usuários que poderão utilizar os filtros da mbrowse).
SXOffice	Cadastro de relacionamento entre as entidades (tabelas) e as consultas TOII.

## **Ambientes e tabelas**

Na aplicação PROTHEUS as tabelas de dados podem ter uma estrutura mais simples e econômica, com tabelas em DBF/ADS do fabricante Extended System ou CTREE do fabricante FAIRCOM ou uma estrutura mais robusta e complexa, em bases SQL (SQLSERVER da Microsoft, ORACLE, DB II da IBM, SYBASE, MYSQL, POSTGREGEE, etc.).

No caso do SQL o acesso é feito através do TOPCONNECT / DBACCESS, que converte os comandos do ADVPL para este ambiente.

Para permitir uma utilização adequada das tabelas de dados do Sistema por cada um dos Ambientes da aplicação ERP, as tabelas foram divididas em grupos denominados “famílias”. Cada módulo pode utilizar uma ou mais famílias de tabelas específicas para suas atividades, e ainda compartilhar informações com outros módulos, através de famílias comuns a todas as operações realizadas no Sistema.

A tabela a seguir demonstra alguns dos módulos que compõe a aplicação ERP PROTHEUS atualmente:

Ambiente	Identificação
SIGAATF	ATIVO FIXO
SIGACOM	COMPRAS
SIGACON	CONTABILIDADE
SIGAEST	ESTOQUE E CUSTOS
SIGAFAT	FATURAMENTO
SIGAFIN	FINANCEIRO
SIGAFIS	LIVROS FISCAIS
SIGAPCP	PLANEJAMENTO E CONTROLE DA PRODUÇÃO
SIGAGPE	GESTÃO DE PESSOAL
SIGAFAS	FATURAMENTO DE SERVIÇOS
SIGAVEI	VEÍCULOS
SIGALOJA	CONTROLE DE LOJAS/AUTOMAÇÃO COMERCIAL
SIGATMK	CALL CENTER
SIGAOFI	OFICINAS
SIGAPON	PONTO ELETRÔNICO
SIGAEIC	EASY IMPORT CONTROL
SIGATCF	TERMINAL
SIGAMNT	MANUTENÇÃO DE ATIVOS
SIGARSP	RECRUTAMENTO E SELEÇÃO DE PESSOAL
SIGAQIE	INSPEÇÃO DE ENTRADA – QUALIDADE
SIGAQMT	METODOLOGIA – QUALIDADE

O nome de cada tabela no Protheus é constituído de seis dígitos que são utilizados para formar a seguinte representação:

F            X            X            E            E            0

Onde:

F	SF	X	Primeiro dígito representa a família, o segundo dígito pode ser utilizado para detalhar ainda mais a família especificada no primeiro nível (subfamília), e o terceiro dígito é a numeração sequencial das tabelas da família, iniciando em “0” e finalizando em “Z”.		
E	E	0	Os dois primeiros dígitos identificam a que empresa as tabelas estão vinculadas, lembrando que a informação de filial está contida nos dados da tabela. O último dígito é fixo em “0”.		

A tabela a seguir demonstra algumas das principais famílias de tabelas utilizadas pela aplicação ERP Protheus:

Família		Descrição
S	-	Tabelas pertencentes ao sistema básico, também chamado Classic
S	A	Cadastros de entidades compartilhadas entre os Ambientes (Clientes, Fornecedores, Bancos entre outros)

Família		Descrição
S	B	Cadastrados dos Ambientes de Materiais (Produtos, Saldos entre outros)
S	C	Arquivos de movimentações diversas, utilizados pelos ambientes de Materiais (Solicitação ao Almoxarifado, Solicitação de Compras, Pedido de Compras, Pedido de Vendas, Ordens de Produção entre outros)
S	D	Arquivos de movimentações de estoque (Itens de notas fiscais de entrada e saída, movimentos internos de estoque entre outros)
S	E	Cadastrados e movimentações do ambiente Financeiro
S	F	Cadastrados e movimentações Fiscais (Cabeçalhos das notas fiscais de entrada e saída, cadastro de tipos de entrada e saída, livros fiscais, entre outros)
S	G	Cadastrados do Ambiente de Planejamento e Controle de Produção
S	H	Movimentos do Ambiente de Planejamento e Controle de Produção
S	I	Cadastrados e movimentos do Ambiente Contábil (descontinuado)
S	N	Cadastrados e movimentos do Ambiente Ativo Fixo
S	R	Cadastrados e movimentos do Ambiente Gestão de Pessoal
S	X	Tabelas de configuração do Sistema
S	Z	Tabelas livres para utilização e projetos específicos em clientes
A	-	Gestão de Projetos
C	-	Contabilidade Gerencial

Família		Descrição
C	T	Contabilidade Gerencial
C	V	Contabilidade Gerencial
C	W	Contabilidade Gerencial
D	-	Transportadoras e derivados
E	-	Comércio exterior e derivados
G	-	Gestão Hospitalar
J	-	Gestão Educacional
N	-	Serviços Públicos
P	-	Reservado para projetos da fábrica de software
Q	-	Qualidade e derivados
R	-	Recursos Humanos e derivados
T	-	Plano de Saúde
W	-	Workflow
Z	-	Tabelas livres para utilização e projetos específicos em clientes em adição à família SZ

## Índices

Cada tabela do Sistema possui seus índices definidos no arquivo de configuração SIX, o qual pode ser atualizado com a utilização do módulo Configurador.

Os arquivos de índices das tabelas de Sistema serão criados de acordo com o banco de dados utilizado (ISAM ou conexão via TOPCONNECT).

Para os bancos de dados ISAM serão gerados arquivos com a mesma nomenclatura da tabela de dados, mas com uma extensão diferenciada (atualmente .CDX). No caso da utilização de um banco de dados, cada índice será uma numeração sequencial em função do nome da tabela original.

As especificações das chaves de índices de cada uma das tabelas está disponível no arquivo de sistema SIX, e a chave única da tabela utilizada para banco de dados está descrita na tabela SX2.

## Menus

Cada módulo da aplicação ERP possui um menu padrão com todas as funcionalidades disponíveis para o Ambiente, menu este definido através de sintaxe XML (arquivos .XNU).

Os menus possuem uma estrutura padrão que permite ao usuário localizar e identificar facilmente cada uma das funcionalidades oferecidas pelo Ambiente.

### **9.3.3.**

### **Acessando o módulo Configurador**

Para executar o módulo Configurador é necessário que a aplicação Protheus Server esteja em execução e através da aplicação Protheus Remote deverá ser informada como programa inicial a opção SIGACFG.



Figura: Parâmetros de inicialização do Sistema

Após a confirmação, a validação do acesso é feita conforme tela ilustrada a seguir:



Figura: Validação de acesso



Figura: Confirmação do acesso ao módulo Configurador

Logo após a sua confirmação do usuário e senha com direito de administrador, será apresentada a tela inicial do configurador, conforme mostra a figura a seguir:

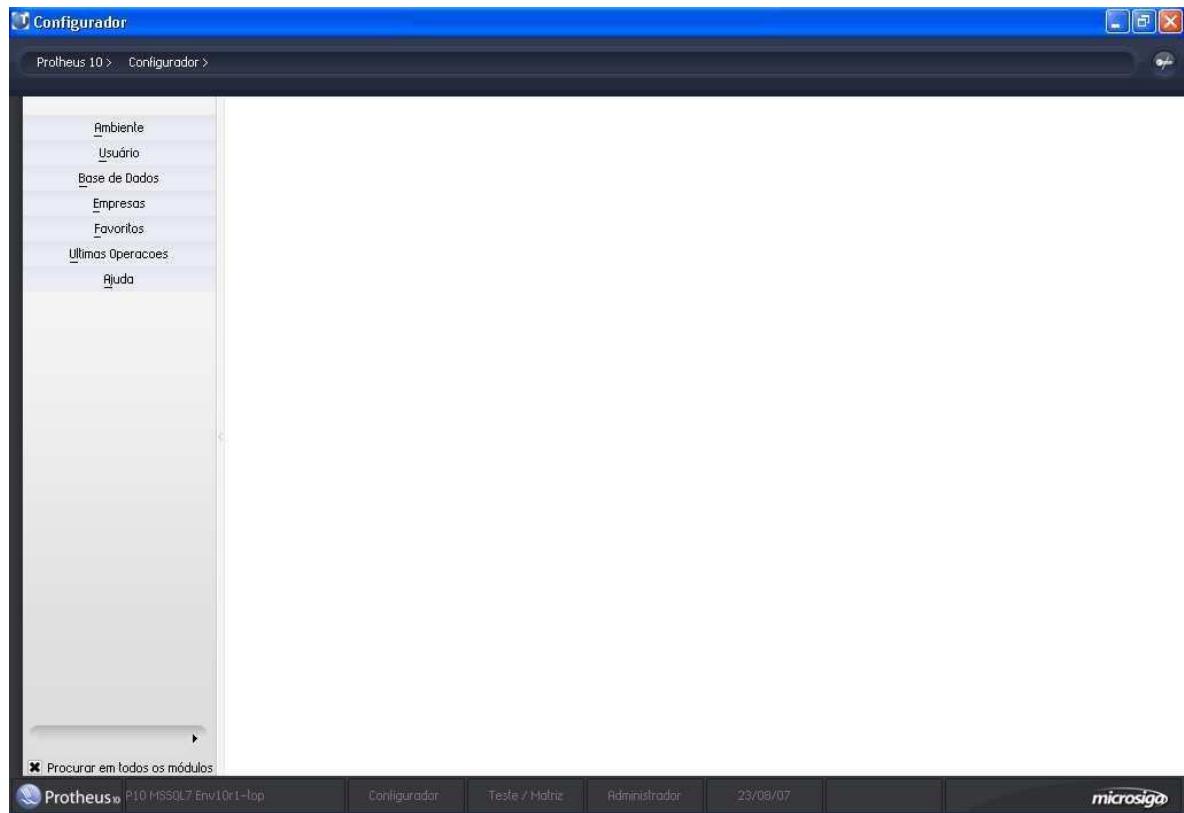


Figura: Interface principal do módulo Configurador

## 9.4. Funcionalidades do Configurador

A customização de um Sistema como o Protheus consiste em adaptar o Sistema de forma a melhor atender as necessidades do cliente.

A flexibilidade de um Sistema, ou seja, sua capacidade de adaptar-se (polimorfismo, aquele que assume várias formas) é uma das mais importantes características de uma solução ERP.

As funcionalidades tratadas pelo Configurador definem a flexibilidade do ERP Protheus. Flexibilizar sem despadronizar, ou seja, tudo que foi customizado permanece válido, mesmo com o desenvolvimento de novas versões.

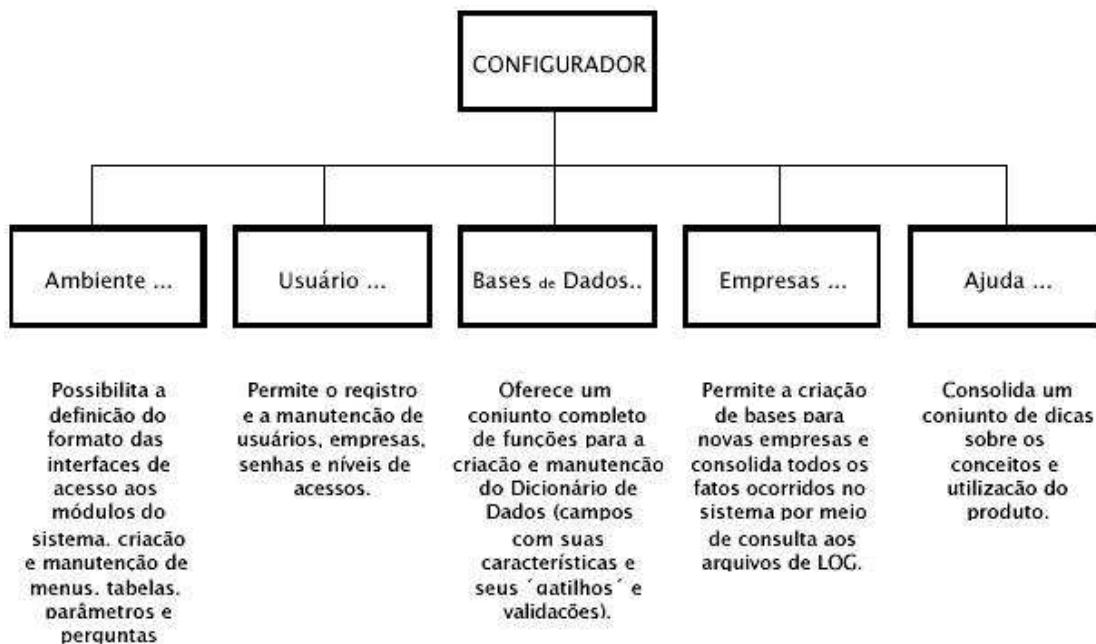


Figura: Principais funcionalidades do módulo Configurador

O Configurador é o programa básico para o processo de customização do Protheus, através da alteração das tabelas da família SX. Neles, o usuário ou o analista de suporte responsável pela implantação configura as informações que serão utilizadas pelos demais ambientes do Sistema.

Essas informações vão de simples parâmetros até complexas expressões e comandos que são interpretados em tempo de execução.

Nos próximos tópicos serão abordadas as funcionalidades de customização disponíveis no Ambiente Configurador, relevantes ao objetivo de desenvolvimento de pequenas customizações para a aplicação ERP.

#### 9.4.1.

#### Dicionário de Dados da aplicação ERP

A idéia do Dicionário de Dados é permitir que o usuário possa incluir ou inibir campos, ou mesmo alterar as propriedades dos campos existentes. Pode, ainda, criar novas tabelas. Ou seja, os programas ao invés de terem os campos definidos em seu código original, lêem o Dicionário em tempo de execução, montando arrays com as propriedades de cada um. A partir daí, sua utilização é normal, através do uso de funções do ADVPL que tornam o trabalho do desenvolvedor transparente a esta arquitetura.

O objetivo do Dicionário de Dados é permitir que o próprio usuário crie novas tabelas ou altere os campos nas tabelas existentes quanto ao seu uso, sua ordem de apresentação, legenda (nos três idiomas), validação, help, obrigatoriedade de preenchimento, inicialização etc.



Figura: Conjunto de pacotes que compõe o Dicionário de Dados

## **9.4.2.**

### **Adição de tabelas ao Dicionário de Dados**

#### **Procedimento**

1. Para adicionar uma tabela ao dicionário de dados de uma empresa, selecione a opção Dicionário de Dados abaixo da empresa que será atualizada. (árvore de opções da parte esquerda da interface visual do Gerenciador de Bases de Dados).
2. Após a seleção da opção Dicionário de Dados serão exibidas as tabelas já cadastradas no arquivo de sistema SX2.

Prefixo	Descrição
AA1	Técnicos
AA2	Habilidades dos Técnicos
AA3	Base Instalada
AA4	Acessórios Da Base Instalada
AA5	Serviços
AA6	Kits de Atendimentos
AA7	Produtos X Ocorrências
AA8	Plano de Manutenção
AA9	Itens do Plano de Manutenção
AAA	Grupos de Cobertura
AAB	Itens do Grupo de Cobertura
AAC	Habilidades Da Amarração
AAD	Índices
AAE	Índices – Taxas
AAF	Históricos
AAG	Ocorrências
AAH	Contrato de Manutenção
AAI	Faq
AAJ	Preventiva
AAK	Obsolescência
AAK	Itens Em Obsolescência
RAM	Contrato Prestação de Serviços
RAA	Itens Prest Serviços Parceria
RAO	Itens Prest Serviços Wms
RAP	Grupo de Atendimento
AB1	Chamado Técnico
AB2	Itens do Chamado Técnico

Figura: Conjunto de tabelas já cadastradas no SX2

3. Após a visualização das tabelas já cadastradas no SX2 da empresa selecionada, utilize

o botão Incluir ( ). Ao utilizar esta opção será exibida a tela para definição dos dados referentes à nova tabela que será criada:



 **Anotações**

---



---



---



---

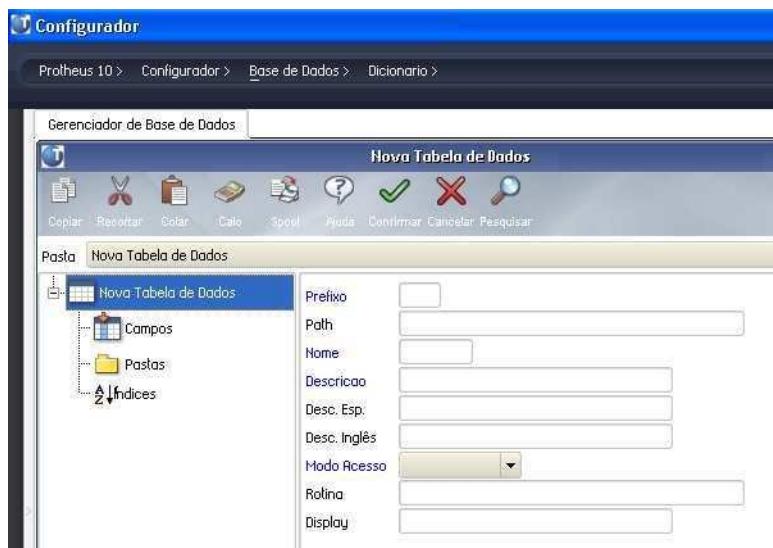


Figura: Cadastro de uma nova tabela

- Realize o preenchimento das informações solicitadas de acordo com as orientações a seguir, e ao término confirme o cadastramento da nova tabela com o botão Confirmar



#### Orientações para o cadastramento de uma nova tabela

---

O domínio SZ1 até SZZ (considerando todos os números e todas as letras no último byte) é reservado para dados exclusivos do usuário, pois esse intervalo não será utilizado pelo Sistema. Caso seja necessário o domínio Z00 a ZZZ também pode ser empregado para desenvolvimentos específicos do cliente.



*Fique  
atento*

Não devem ser criadas tabelas específicas de clientes com quaisquer outras nomenclaturas, o que pode afetar diretamente um processo de atualização futuro.

O nome da tabela é preenchido automaticamente, adicionando 990. Esse dado refere-se à empresa 99 (Teste Matriz) a qual está sendo adicionado à tabela.

O Path refere-se à pasta que conterá efetivamente os dados das tabelas, quando ISAM, As versões com banco de dados relacional não são utilizadas. Essa pasta será criada dentro da pasta indicada na configuração do Sistema como ROOTPATH.

O modo de acesso compartilhado indica que o Sistema possibilitará o uso simultâneo da tabela por duas ou mais filiais. Se for compartilhado o campo Filial fica em branco. Se for exclusivo, grava-se o código da filial ativa e somente ela tem acesso ao registro.

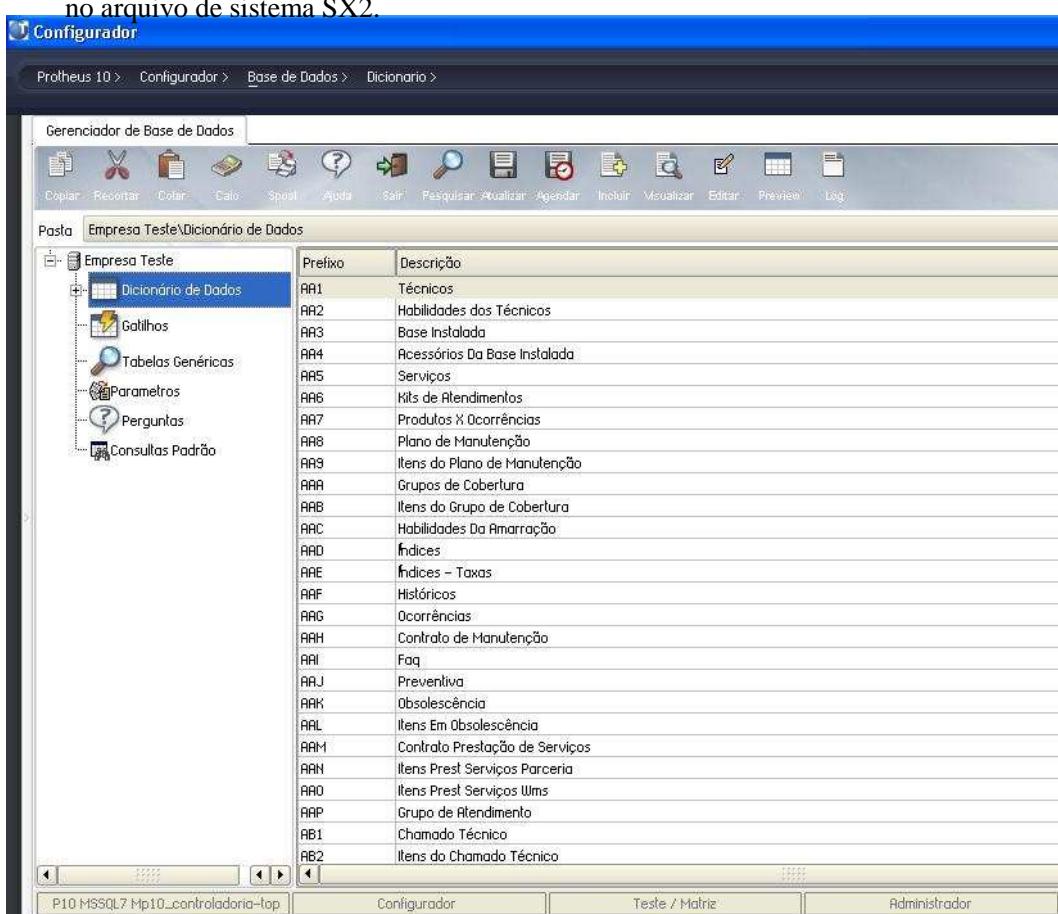
Após a confirmação, a tabela criada passa a fazer parte do cadastro do Dicionário de Dados, contendo somente o campo FILIAL, o qual é criado como padrão pela funcionalidade do módulo.

### **9.4.3.**

### **Adição de campos às tabelas do Dicionário de Dados**

#### **Procedimento**

1. Para adicionar um campo a uma tabela do dicionário de dados de uma empresa, selecione a opção Dicionário de Dados abaixo da empresa que será atualizada. (árvore de opções da parte esquerda da interface visual do Gerenciador de Bases de Dados).
2. Após a seleção da opção Dicionário de Dados serão exibidas as tabelas já cadastradas no arquivo de sistema SX2.



**Figura: Conjunto de tabelas já cadastradas no SX2**

3. Após a visualização das tabelas já cadastradas no SX2 da empresa selecionada,  ( ). Ao

localize e selecione a tabela que será atualizada, e utilize o botão Editar (  ). Utilizar esta opção será exibida a tela de manutenção de campos da tabela selecionada:



Figura: Estrutura de uma tabela já cadastrada no Sistema

4. Selecione a opção Campos (  ), para que sejam exibidos os campos disponíveis para a tabela no arquivo de sistema SX3.

**Configurador**

Pretheus 10 > Configurador > Base de Dados > Dicionário >

Gerenciador de Base de Dados

**Editar Tabela de Dados - SAI**

Copiar Recortar Colar Spool Ayuda Confirmar Cancelar Pesquisar Preview Visualizar Incluir Editar Excluir Reservado

Pasta Clientes\Campos

	Ordem	Campo	Título	Descrição
1	1	A1_FILIAL	Filial	Filial do Sistema
2	2	A1_COD	Código	Código do Cliente
3	3	A1_LOJA	Loja	Loja do Cliente
4	4	A1_PESSOA	Física/Jurídica	Pessoa Física/Jurídica
5	5	A1_NOME	Nome	Nome do cliente
6	6	A1_REDUCOZ	N Fantasia	Nome Reduzido do cliente
7	7	A1_TIPO	Tipo	Tipo do Cliente
8	8	A1_END	Endereço	Endereço do cliente
9	9	A1_MUN	Município	Município do cliente
10	10	A1_EST	Estado	Estado do cliente
11	11	A1_NATUREZA	Natureza	Código da Nat Financeira
12	12	A1_ESTADO	Nome Estado	Nome do Estado Fornecedor
13	13	A1_BAIRRO	Bairro	Bairro do cliente
14	14	A1_CEP	CEP	Cod Endereçamento Postal
15	15	A1_DDI	DDI	Código do DDI
16	16	A1_DDD	DDD	Código do DDD
17	17	A1_TEL	Telefone	Telefone do cliente
18	18	A1_TELEX	Telex	Telex do cliente
19	19	A1_FAX	FAX	Número do FAX do cliente
20	20	A1_ENDCOB	End.Cobrança	End.de cobr. do cliente
21	21	A1_PAIS	País	Código do País
22	22	A1_PAISDES	Descr. País	Descr. País
23	23	A1_ENDENT	End.Entrega	End.de entr. do cliente

Pode alterar a ordem Não pode alterar a ordem

P10 MSSQL7 Mp10...controladoria-top | Configurador | Teste / Matriz | Administrador

Figura: Estrutura de campos de uma tabela já cadastrada no Sistema

5. Após a visualização dos campos já cadastrados no SX3 da tabela selecionada, utilize a opção Incluir ( ). Ao utilizar esta opção será exibida a tela para definição dos dados referentes ao novo campo que será criado:

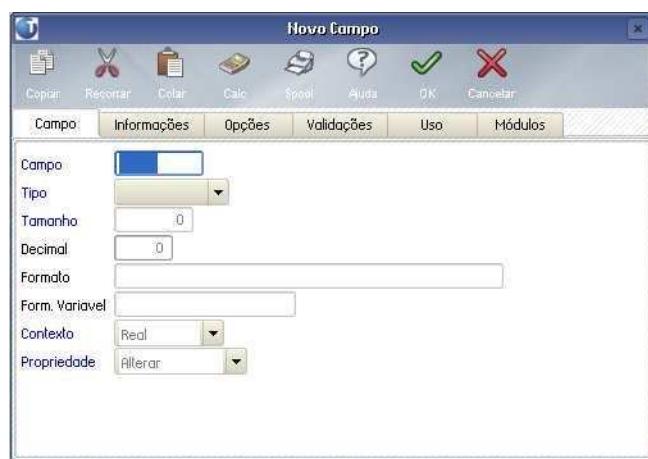


Figura: Dados para parametrização de um novo campo no Sistema

5. Realize o preenchimento das informações solicitadas de acordo com as orientações a seguir, e ao término confirme o cadastramento do novo campo para a tabela com o

botão Confirmar (  ).

6. Confirme as atualizações para a tabela selecionada com o botão Confirmar (  ).

7. Atualize as configurações do Sistema com o botão Atualizar (  ).

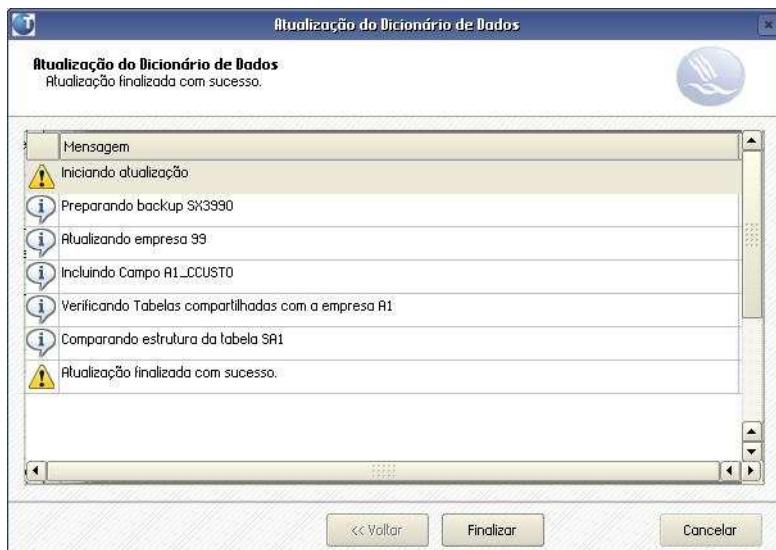


Figura: Atualização dos dados do Sistema

---

### Orientações para o cadastramento de um novo campo

1. As propriedades definidas no Dicionário de Dados (SX3) são as seguintes:

Nome do campo: Todos os campos têm como prefixo o próprio nome da tabela, sendo que para tabelas da família “S”, o prefixo do campo é composto apenas pelos próximos dois dígitos. No caso das demais tabelas, o prefixo do campo serão os três dígitos identificadores da tabela.

**Tipo do campo:** Indica se é caractere, numérico, lógico, data ou memo. É claro que a mudança do tipo de campo deve ser feita com muito cuidado, pois, se tivermos um campo numérico usado em cálculos e ele for alterado para caractere, certamente teremos um erro.

**Tamanho do campo:** Também aqui é necessário certo cuidado ao alterá-lo, pois poderemos ter truncamentos em relatórios e consultas em que há espaço para conteúdos maiores que o original.

**Formato de edição:** Define como o campo aparece nas telas e nos relatórios.

**Contexto:** Pode ser real ou virtual. O contexto virtual cria o campo somente na memória e não na tabela armazenada no disco. Isso é necessário porque os programas de cadastramento e de consulta genérica apresentam somente uma tabela de cada vez. Assim, se quisermos apresentar um campo de uma outra tabela, ou mesmo o resultado de um cálculo, sem que tal informação ocupe espaço físico no HD, utilizamos o contexto virtual. Campos virtuais normalmente são alimentados por gatilhos.

**Propriedade:** Indica se um campo pode ou não ser alterado pelo usuário. Exemplo: saldos normalmente não podem, pois quem cuida dessa tarefa são os programas.

## 2. Demais características que devem ser observadas na configuração do campo:



### Guia: Campo

O campo Decimal será solicitado somente para os campos de tipo numérico.

O formato “!” indica que o caractere será sempre maiúsculo, independente da ação do usuário. O formato “@!” indica que essa característica estende-se por todo o campo.

O contexto real indica que o campo existirá efetivamente no banco de dados e o contexto virtual significa que o campo existirá apenas no dicionário de dados e não fisicamente.

A propriedade alterar indica que o campo pode ser alterado.

Nesta janela, os dados estão classificados em seis pastas com objetivos de preenchimento bem específicos:

#### Guia: Informações

Contém as informações a respeito dos títulos.

Título: É a legenda que aparece nas telas/relatórios. Há inclusive três campos para esta finalidade: em português, espanhol e inglês. Esta propriedade pode ser alterada à vontade, pois não interfere em nenhum processamento.

Descrição e Help: São propriedades que objetivam documentar o campo.

#### Guia: Opções

Contém os dados que facilitam a digitação.

#### Guia: Validações

Representam as regras de validação do campo.

Validações: Nesta propriedade, escreve-se uma função de validação do campo que está sendo digitado. Existe um conjunto de funções disponíveis no ADVPL apropriadas para esse caso.

Todas as validações informadas serão executadas no momento do preenchimento do próprio campo. Uma validação pode ser uma expressão lógica ou uma função de usuário que retorna um valor lógico Verdadeiro ou Falso. O sistema só permitirá o avanço para o próximo campo quando o respectivo preenchimento resultar Verdadeiro, seja na expressão ou no retorno da função.

#### Guia: Uso

Descreve a forma de utilização do campo.

## Guia: Módulos

Relaciona todos os módulos em que o campo será utilizado.



Anotações

---

---

---

---

#### **9.4.4.**

#### **Adição de índices para as tabelas do Dicionário de Dados**

Conforme mencionado anteriormente, no Ambiente Protheus uma tabela pode ter vários índices, os quais serão gerados de acordo com o banco de dados configurado para o Sistema.

Os índices do Sistema auxiliam na seleção e obtenção de informações da base de dados, além de determinar a ordem de apresentação dos registros de uma tabela em consultas e relatórios.

##### **Procedimento**

1. Para adicionar um índice a uma tabela do dicionário de dados de uma empresa, selecione a opção Dicionário de Dados abaixo da empresa que será atualizada. (árvore de opções da parte esquerda da interface visual do Gerenciador de Bases de Dados).
  
2. Após a seleção da opção Dicionário de Dados serão exibidas as tabelas já cadastradas no arquivo de sistema SX2.

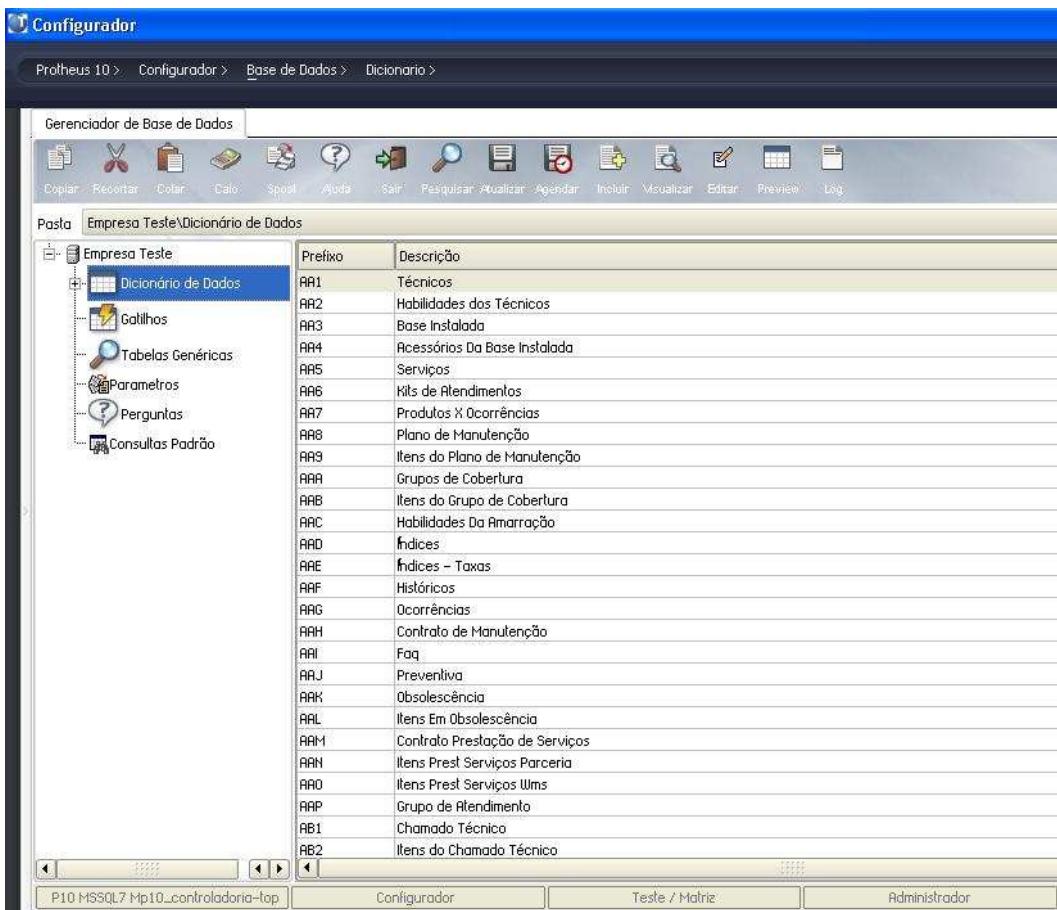


Figura: Conjunto de tabelas já cadastradas no SX2

3. Após a visualização das tabelas já cadastradas no SX2 da empresa selecionada,



localize e selecione a tabela que será atualizada, e utilize o botão **Editar**). Ao utilizar esta opção será exibida a tela de manutenção de campos da tabela selecionada:

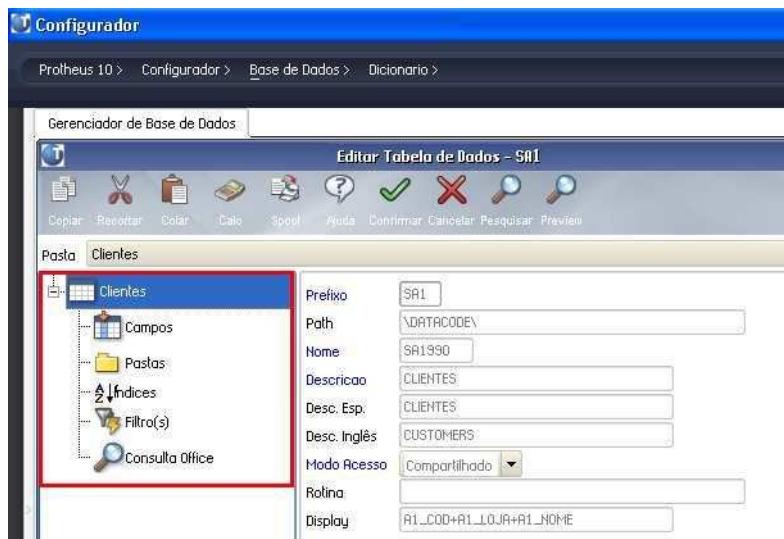


Figura: Estrutura de uma tabela já cadastrada no Sistema

4. Selecione a opção índices ( ), para que sejam exibidos os índices disponíveis para a tabela no arquivo de sistema SIX.

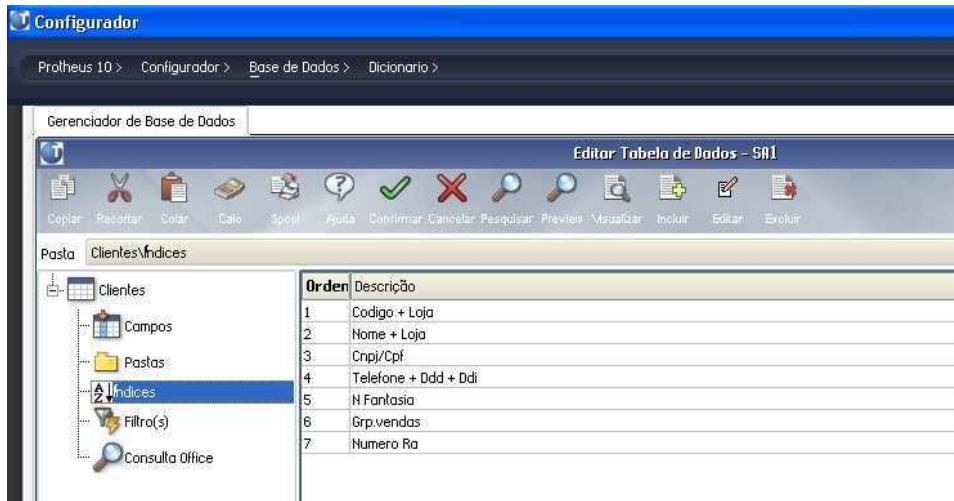


Figura: Índices disponíveis para a tabela no arquivo de sistema SIX

5. Após a visualização dos índices já cadastrados no SIX para a tabela selecionada, utilize



a opção Incluir (incluir). Ao utilizar esta opção, será exibida a tela para definição dos dados referentes ao novo índice que será criado:



Figura: Adição de um índice para uma tabela

6. Realize o preenchimento das informações solicitadas de acordo com as orientações a seguir, e ao término confirme o cadastramento do novo índice para a tabela com o botão

Confirmar ( ).

7. Confirme as atualizações para a tabela selecionada com o botão Confirmar ( ).



8. Atualize as configurações do Sistema com o botão Atualizar ( ).

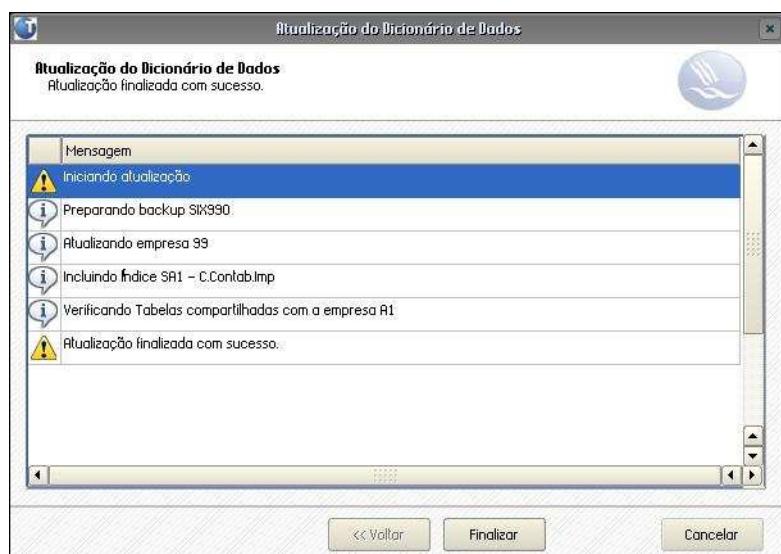


Figura: Atualização dos dados do Sistema

## **Orientações para o cadastramento de um novo índice**

O Nickname é uma identificação complementar do índice o qual pode ser utilizada para auxiliar o desenvolvedor na utilização desta ordem em uma aplicação, a qual pode ser padrão do Sistema ou específica de um cliente.

Para selecionar os campos já cadastrados na tabela, pode ser utilizado o botão



Campos ( ). Esta facilidade preenche, automaticamente, os campos de descrição.

O campo relativo à filial sempre faz parte dos índices, com exceção do SM2, para que os registros nas tabelas estejam agrupados por filiais, independente desta tabela ser compartilhada entre as filiais.

Uma tabela poderá ter vários índices cadastrados no Dicionário de Dados. Em determinado momento, porém, apenas um deles oferecerá acesso ao registro. Essa ordem pode ser alterada em tempo de execução pelos programas da aplicação, através do comando DBSetOrder(), ou através da definição de uma ordem específica na utilização de queries para acesso aos dados diretamente em bancos de dados de Ambientes TOPCONNECT (DbAcess).

### Procedimento

1. Para adicionar um gatilho a um campo de uma tabela do dicionário de dados de uma empresa, selecione a opção Gatilho abaixo da empresa que será atualizada. (árvore de opções da parte esquerda da interface visual do Gerenciador de Bases de Dados).
2. Após a seleção da opção Gatilhos serão exibidos os itens já cadastradas no arquivo de sistema SX7.

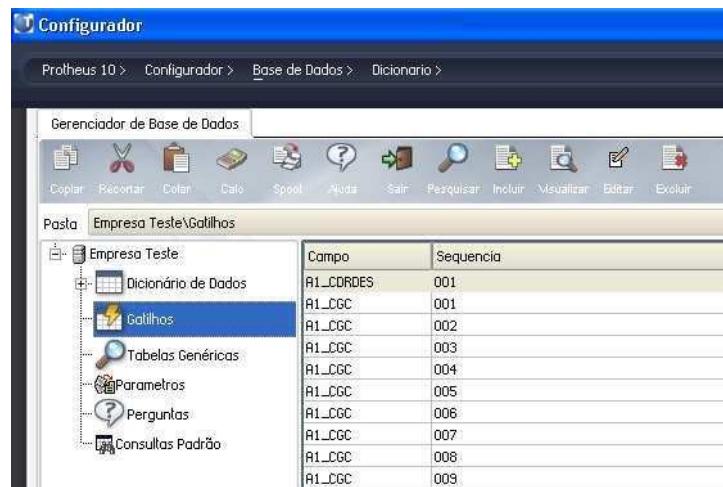


Figura: Conjunto de gatilhos já cadastrados no SX7

3. Após a visualização dos gatilhos já cadastrados no SX7 da empresa selecionada, utilize

o botão Incluir (  ) para realizar o cadastro de um novo gatilho no Sistema:

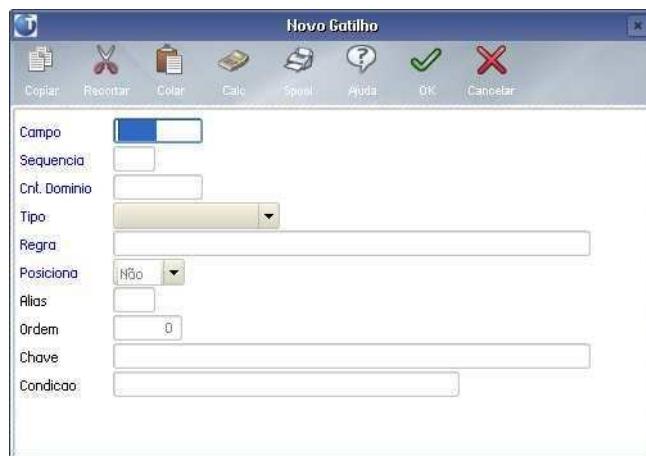


Figura: Dados para o cadastro de um novo gatilho no Sistema

4. Realize o preenchimento das informações solicitadas de acordo com as orientações a seguir, e ao término confirme o cadastramento do novo gatilho de sistema com o

botão Confirmar (  ).

#### Orientações para o cadastramento de um novo gatilho

Pode haver vários gatilhos para o mesmo campo. A ordem de execução é determinada pelo campo Sequência.

Os tipos do Gatilho Primário, Estrangeiro e de Posicionamento definem se o Contra Domínio é um campo da mesma tabela, de outra tabela ou se o gatilho deve realizar um posicionamento, respectivamente.

A regra pode ser uma expressão que resulta em um valor a ser preenchido no Contra Domínio.

O posicionamento igual a Sim indica que será executado um comando de busca do registro de acordo com a chave indicada.

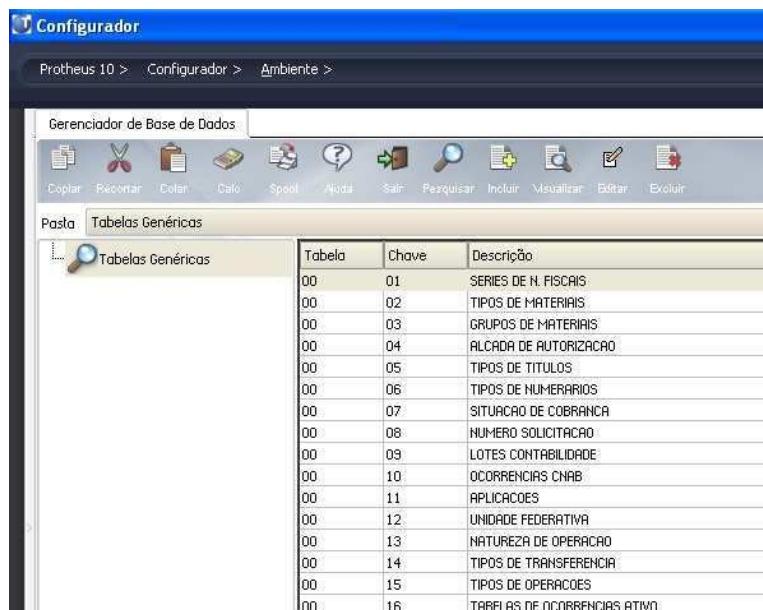
O Alias, a Ordem e a Chave descrevem a tabela envolvida no gatilho, seu índice e a chave para que a funcionalidade se posicione no registro adequado.

## **9.4.6.**

### **Criação de Tabelas Genéricas**

#### Procedimento

1. Para adicionar uma tabela genérica, selecione os menus Ambiente, Cadastros, Tabelas.
2. Após a seleção da opção Tabelas serão exibidos os itens já cadastradas no arquivo de sistema SX5.



The screenshot shows a software interface titled 'Configurador' with a blue header bar. Below it, the path 'Protheus 10 > Configurador > Ambiente >' is visible. The main window is titled 'Gerenciador de Base de Dados' and contains a toolbar with various icons for copy, paste, cut, find, etc. Below the toolbar, there are two tabs: 'Pasta' and 'Tabelas Genéricas'. The 'Tabelas Genéricas' tab is selected, displaying a list of 16 generic tables. The table has columns: 'Tabela' (number), 'Chave' (key), and 'Descrição' (description). The descriptions are as follows:

Tabela	Chave	Descrição
00	01	SERIES DE N. FISCAIS
00	02	TIPOS DE MATERIAIS
00	03	GRUPOS DE MATERIAIS
00	04	ALCADA DE AUTORIZACAO
00	05	TIPOS DE TITULOS
00	06	TIPOS DE NUMERARIOS
00	07	SITURCAO DE COBRANCA
00	08	NUMERO SOLICITACAO
00	09	LOTES CONTABILIDADE
00	10	OCCORENCIAS CNAB
00	11	APLICACOES
00	12	UNIDADE FEDERATIVA
00	13	NATUREZA DE OPERACAO
00	14	TIPOS DE TRANSFERENCIA
00	15	TIPOS DE OPERACOES
00	16	TABELAS DE OCCORENCIAS ATIVO

Figura: Conjunto de Tabelas já cadastradas no SX5

3. Após a visualização das tabelas já cadastrados no SX5 da empresa selecionada, utilize



o botão Incluir ( ) para realizar o cadastro de uma nova tabela no Sistema:

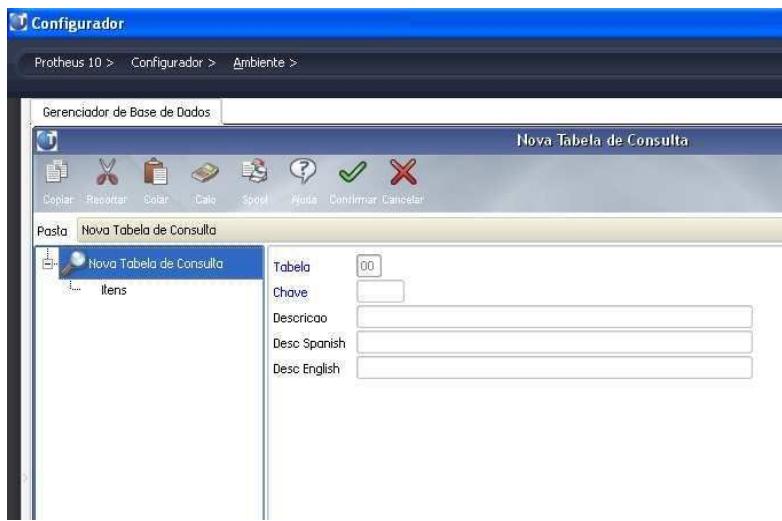
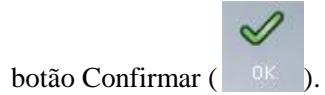


Figura: Dados para o cadastro de uma nova tabela no Sistema

4. Realize o preenchimento das informações solicitadas de acordo com as orientações a seguir, e ao término confirme o cadastramento da nova tabela do sistema com o



## 9.4.7.

## Criação de Parâmetros

### Procedimento

1. Para adicionar um Parâmetro, selecione os menus Ambiente, Cadastros, Parâmetros.
2. Após a seleção da opção Tabelas serão exibidos os itens já cadastradas no arquivo de sistema SX6.

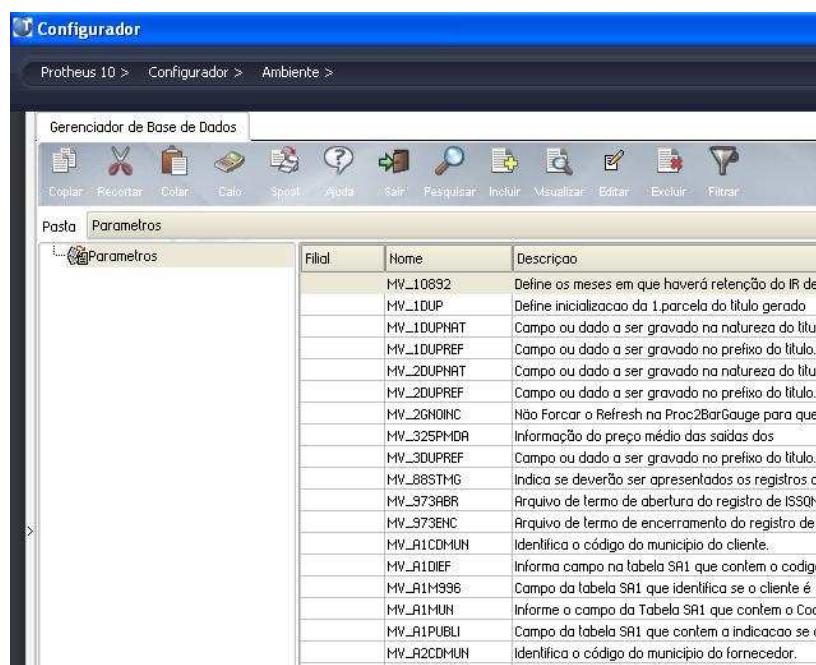


Figura: Conjunto de Parâmetros já cadastrados no SX6

3. Após a visualização dos Parâmetros já cadastrados no SX6 da empresa selecionada, utilize o botão Incluir ( ) para realizar o cadastro de uma nova tabela no sistema:



Figura: Dados para o cadastro de um novo parâmetro no Sistema

4. Realize o preenchimento das informações solicitadas de acordo com as orientações a seguir, e ao término confirme o cadastramento do novo Parâmetro do Sistema com o

botão Confirmar ( ).

## **10. TOTVS DEVELOPMENT STUDIO**

A ferramenta TOTVS Development Studio é um programa que faz parte do Protheus e permite o trabalho de edição, compilação e depuração de programas escritos em ADVPL.

### Projeto

Um programa para ser compilado deve ser vinculado a um projeto. Normalmente, programas que fazem parte de um determinado módulo ou ambiente estão em um mesmo projeto.

A vinculação dos programas a um projeto é feita por meio dos arquivos do tipo PRW. Na verdade, um projeto pode ser constituído de um ou mais arquivos deste tipo, que por sua vez, podem ter uma ou mais funções, conforme ilustra o diagrama a seguir:

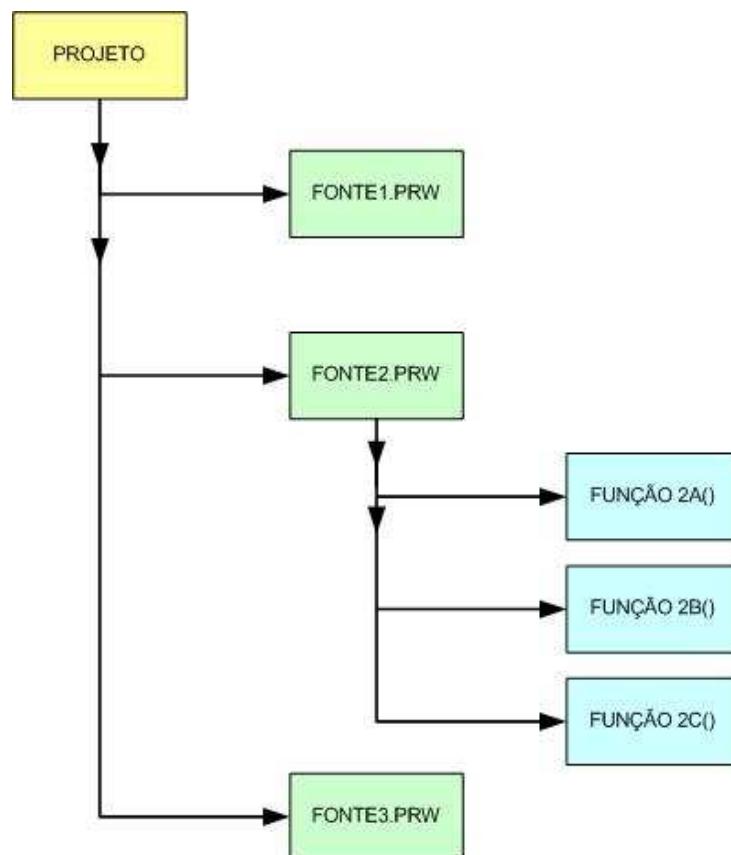


Figura: Representação da estrutura de um projeto no DEV-Studio

## Compilação

Uma vez adicionado a um projeto e compilado sem incidências de erros de código, o objeto resultante será cadastrado no RPO (Repositório de Objetos) e poderá ser utilizado pela aplicação ERP.

A compilação dos itens de um projeto pode ser realizada individualmente, por grupo de fontes (pastas) ou ainda selecionando-o inteiramente. Cada um dos fontes será processado e compilado separadamente, permitindo a visualização do progresso da operação e das mensagens de aviso (warnings) ou erros (critical errors) na guia Mensagens.

## Execução

Para que os objetos compilados e disponíveis n RPO sejam utilizados, devem ser observadas as seguintes regras:

Se o programa não manipula arquivos, pode-se chamá-lo diretamente do DEV-Studio (nome no lado direito da barra de ferramentas);

Se o programa manipula tabelas existem duas opções:

Adicionar o programa no menu de um dos Ambientes e executá-lo através do Remote.

Realizar a preparação do Ambiente na própria rotina, permitindo sua execução diretamente pelo DEV-Studio.

Não se pode compilar um programa com o Remote e o Monitor abertos, tenha este finalizado ou não por erro.

## Análise e depuração de erros

Para identificar as causas de erros, a ferramenta DEV-Studio possui diversos recursos que auxiliam o DEBUG.

A ação de DEBUG necessita que o programa seja executado a partir do DEV-Studio, sendo necessário observar as seguintes regras:

Definir e marcar os pontos de parada mais adequados a análise do fonte;

Executar a rotina através do DEV-Studio, selecionando seu nome diretamente, ou o módulo que contém a opção ou a ação que a executará;

A partir do momento em que o DEV-Studio pausar o processamento em um dos pontos de parada especificados previamente, podem ser utilizadas as janelas de visualização, disponíveis no DEV-Studio, que são:

- Variáveis Locais
- Variáveis Privates
- Variáveis Public's
- Variáveis Static's
- Janela da Watch's
- Janela de Tabelas e Campos
- Pilha de Chamadas

Através da Janela de Watch's é possível determinar quais variáveis devem ser exibidas;

Na pilha de chamadas, verifica-se a sequência de chamadas das funções;

Na pasta de Comandos, pode-se, enquanto o programa estiver pausado, escrever qualquer comando e ao dar Enter, ele é executado, permitindo pesquisar palavras e expressões no próprio fonte ou em qualquer fonte armazenado no HD;

Ao parar pode-se ou continuar o programa até o próximo ponto de parada, caso haja um outro definido, ou executar linha a linha da rotina.

## Interface da aplicação

Por ser um ambiente integrado de desenvolvimento, o DEV-Studio proporciona todas essas facilidades, por meio de interface única como ilustra a figura a seguir:

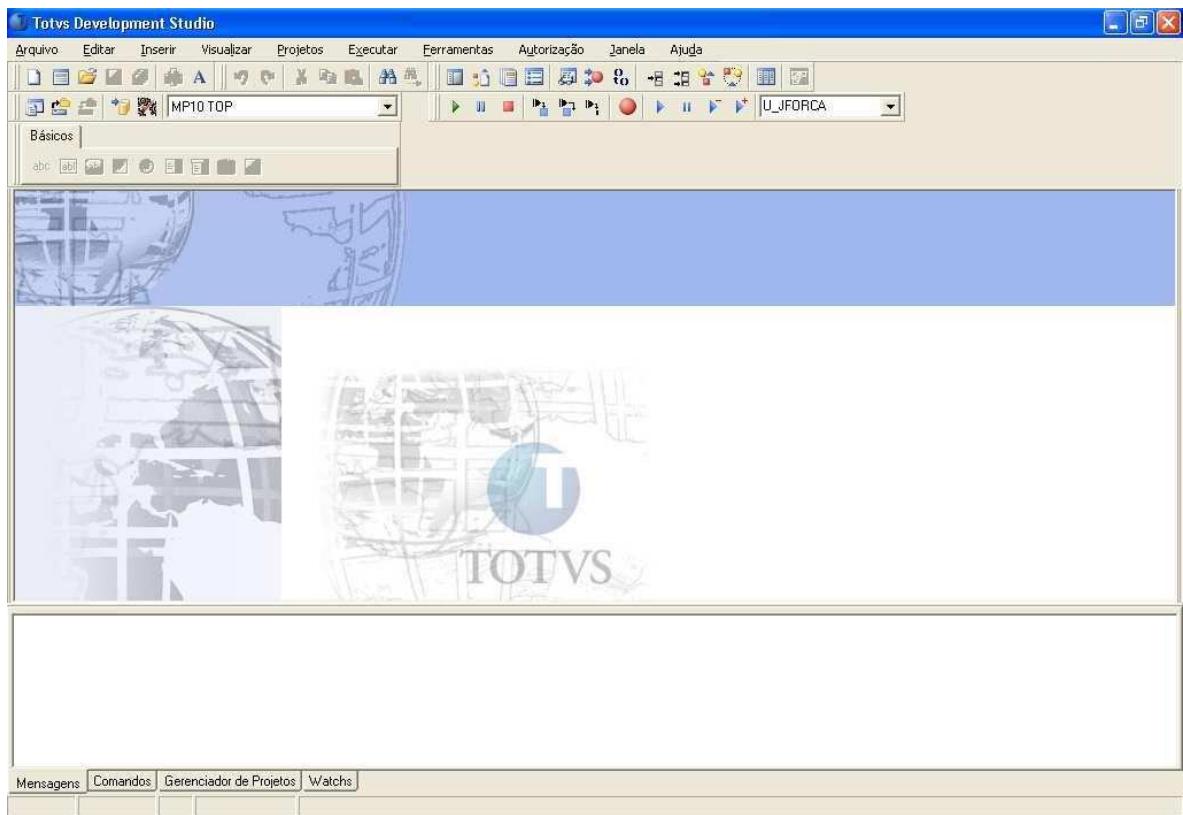


Figura: Interface principal do TOTVS Development Studio

O DEV-Studio apresenta, no topo da tela, um conjunto de opções de menu e uma série de botões que facilitam a sua manipulação.

Na tela central, é apresentado o código das funções em ADVPL. Na parte inferior são exibidas algumas pastas que facilitam a execução de comandos, exibição de conteúdos de variáveis e mensagens, bem como dados sobre o projeto.

## **DESENVOLVIMENTO DE PEQUENAS CUSTOMIZAÇÕES**

### **11. ACESSO E MANIPULAÇÃO DE BASES DE DADOS EM ADVPL**

---

Como a linguagem ADVPL é utilizada no desenvolvimento de aplicação para o sistema ERP Protheus, ela deve possuir recursos que permitam o acesso e a manipulação de informações, independentemente da base de dados para o qual o Sistema foi configurado.

Desta forma a linguagem possui dois grupos de funções distintos para atuar com os bancos de dados:

Funções de manipulação de dados genéricos;

Funções de manipulação de dados específicas para ambientes TOPCONNECT / DBACCESS.

#### **Funções de manipulação de dados genéricos**

---

As funções de manipulação de dados, ditos como genéricos, permitem que uma aplicação ADVPL seja escrita da mesma forma, independente se a base de dados configurada para o sistema ERP for do tipo ISAM ou padrão SQL.

Muitas destas funções foram inicialmente herdadas da linguagem CLIPPER, e mediante novas implementações da área de Tecnologia da Microsiga foram melhoradas e adequadas às necessidades do ERP. Por esta razão é possível encontrar em documentações da linguagem CLIPPER informações sobre funções de manipulação de dados utilizados na ferramenta ERP.

Dentre as melhorias implementadas pela área de Tecnologia da Microsiga, podemos mencionar o desenvolvimento de novas funções como, por exemplo, a função MsSeek() - versão da Microsiga para a função DbSeek(), e a integração entre a sintaxe ADVPL convencional e a ferramenta de acesso a bancos de dados no padrão SQL – TOPCONNECT (DbAccess).

A integração, entre a aplicação ERP e a ferramenta TOPCONNECT, permite que as funções de acesso e manipulação de dados escritos em ADVPL sejam interpretados e convertidos para uma sintaxe compatível com o padrão SQL ANSI e desta forma aplicados aos SGDBs (Sistemas Gerenciadores de Bancos de Dados) com sua sintaxe nativa.

## Funções de manipulação de dados para Ambientes TOPCONNECT / DBACCESS

Para implementar um acesso mais otimizado e disponibilizar no Ambiente ERP funcionalidades que utilizem de forma mais adequada os recursos dos SGDBs homologados para o Sistema, foram implementadas funções de acesso e manipulação de dados específicas para Ambientes TOPCONNECT/DBACCESS.

Estas funções permitem que o desenvolvedor ADVPL execute comandos em sintaxe SQL, diretamente de um código fonte da aplicação, disponibilizando recursos como execução de queries de consulta, chamadas de procedures e comunicação com outros bancos de dados através de ODBC's.



*Fique  
atento*

As funções específicas para Ambientes TOPCONNECT serão abordadas no material de ADVPL Avançado.

## **11.1. Diferenças e compatibilizações entre bases de dados**

Como a aplicação ERP pode ser configurada para utilizar diferentes tipos de bases de dados é importante mencionar as principais diferenças entre estes recursos, o que pode determinar a forma como o desenvolvedor optará por escrever sua aplicação.

### Acesso a dados e índices

No acesso a informações, em bases de dados do padrão ISAM, são sempre lidos os registros inteiros, enquanto no SQL pode-se ler apenas os campos necessários naquele processamento.

O acesso direto é feito através de índices que são tabelas paralelas às tabelas de dados e que contêm a chave e o endereço do registro, de forma análoga ao índice de um livro. Para cada chave, é criado um índice próprio.

Nas bases de dados padrão ISAM os índices são armazenados em um único arquivo do tipo CDX, já nos bancos de dados padrão SQL cada índice é criado com uma numeração sequencial, tendo como base o nome da tabela a qual ele está relacionado.

A cada inclusão ou alteração de um registro todos os índices são atualizados, tornando necessário planejar adequadamente quais e quantos índices serão definidos para uma tabela, pois uma quantidade excessiva pode comprometer o desempenho dessas operações.

Deve ser considerada a possibilidade da utilização de índices temporários para os processos específicos, os quais serão criados em tempo de execução da rotina. Esse fator deve levar em consideração o “esforço” do Ambiente a cada execução da rotina e a periodicidade com a qual é executada.

### Estrutura dos registros (informações)

Nas bases de dados padrão ISAM cada registro possui um identificador nativo ou ID sequencial e ascendente que funciona como o endereço base daquela informação.

Este ID, mais conhecido como RECNO ou RECNOMBRE é gerado no momento de inclusão do registro na tabela e somente será alterado se a estrutura dos dados dessa tabela sofra alguma manutenção. Dentre as manutenções que uma tabela de dados ISAM pode sofrer, é possível citar a utilização do comando PACK, o qual apagará fisicamente os registros deletados da tabela, forçando uma renumeração dos identificadores de todos os registros. Essa situação também torna necessária a recriação de todos os índices vinculados àquela tabela.

Isso ocorre nas bases de dados ISAM devido ao conceito de exclusão lógica de registros que as mesmas possuem. Já os bancos de dados padrão SQL nativamente utilizam apenas o conceito de exclusão física de registros, o que para outras aplicações seria transparente, mas não é o caso do ERP Protheus.

Para manter a compatibilidade das aplicações desenvolvidas para as bases de dados padrão ISAM, a área de Tecnologia e Banco de Dados da Microsiga implementou, nos bancos de dados padrão SQL, o conceito de exclusão lógica de registros existente nas bases de dados ISAM, por meio da criação de campos de controle específicos: R\_E\_C\_N\_O\_, D\_E\_L\_E\_T\_ e R\_E\_C\_D\_E\_L.

Esses campos permitem que a aplicação ERP gerencie as informações do banco de dados, da mesma forma que as informações em bases de dados ISAM.

Com isso o campo R\_E\_C\_N\_O\_ será um identificador único do registro dentro da tabela, funcionando como o ID ou RECNUMBER de uma tabela ISAM, mas utilizando um recurso adicional disponível nos bancos de dados relacionais conhecido com Chave Primária.

Para a aplicação ERP Protheus o campo de controle R\_E\_C\_N\_O\_ é definido em todas as tabelas como sendo sua chave primária, o que transfere o controle de sua numeração sequencial ao banco de dados.

O campo D\_E\_L\_E\_T\_ é tratado internamente pela aplicação ERP como um “flag” ou marca de exclusão. Dessa forma, os registros que estiverem com este campo marcado serão considerados como excluídos logicamente. A execução do comando PACK, em uma tabela de um banco de dados padrão SQL, visa excluir fisicamente os registros com o campo D\_E\_L\_E\_T\_ marcado, mas não causará o efeito de renumeração de RECNO (no caso R\_E\_C\_N\_O\_) que ocorre na tabela de bases de dados ISAM.

## **11.2. Funções de acesso e manipulação de dados**

As funções de acesso e manipulação de dados, descritas neste tópico, são as classificadas anteriormente como funções genéricas da linguagem ADVPL, permitindo que as mesmas sejam utilizadas independentemente da base de dados para a qual a aplicação ERP está configurada.

As funções de acesso e manipulação de dados definem basicamente:

- Tabela que está sendo tratada;
- Campos que deverão ser lidos ou atualizados;
- Método de acesso direto às informações (registros e campos).

Dentre as funções ADVPL disponíveis para acesso e manipulação de informações, este material detalhará as seguintes opções:

SELECT()  
DBSELECTAREA()  
DBSETORDER()  
DBSEEK() E MSSEEK()  
DBSKIP()  
DBGOTO()  
DBGOTOP()  
DBGOBOTTOM()  
DBSETFILTER()  
RECLOCK()  
SOFTLOCK()  
MSUNLOCK()  
DBDELETE()  
DBUSEAREA()  
DBCLOSEAREA()  
DBRLOCK()

Sintaxe	<code>DBRLOCK(xIdentificador)</code>
Descrição	<p>Função de base de dados, que efetua o lock (travamento) do registro identificado pelo parâmetro <code>xIdentificador</code>. Esse parâmetro pode ser o <code>Recno()</code> para tabelas em formato ISAM, ou a chave primária para bancos de dados relacionais.</p> <p>Se o parâmetro <code>xIdentificador</code> não for especificado, todos os locks da área de trabalho serão liberados e o registro posicionado será travado e adicionado em uma lista de registros bloqueados.</p>

#### DBCLOSEAREA()

Sintaxe	<code>DbCloseArea()</code>
Descrição	Permite que um “alias” presente na conexão seja fechado, o que viabiliza novamente seu uso em outro operação. Este comando tem efeito apenas no alias ativo na conexão, sendo necessária sua utilização em conjunto com o comando <code>DbSelectArea()</code> .

#### DBCOMMIT()

Sintaxe	<code>DBCOMMIT()</code>
Descrição	Efetua todas as atualizações pendentes na área de trabalho ativa.

#### DBCOMMITALL()

Sintaxe	<code>DBCOMMITALL()</code>
Descrição	Efetua todas as atualizações pendentes em todas as áreas de trabalho, em uso pela thread (conexão) ativa.

#### DBDELETE()

Sintaxe	<code>DbDelete()</code>
---------	-------------------------

Descrição	Efetua a exclusão lógica do registro posicionado na área de trabalho ativa, sendo necessária sua utilização em conjunto com as funções RecLock() e MsUnLock().
-----------	--

### DBGOTO()

---

Sintaxe	DbGoto(nRecno)
Descrição	Move o cursor da área de trabalho ativa para o record number (recno) especificado, realizando um posicionamento direto, sem a necessidade de uma busca (seek) prévia.

## **DBGOTOP()**

---

Sintaxe	DbGoTop()
Descrição	Move o cursor da área de trabalho ativa para o primeiro registro lógico.

## **DBGOBOTTON()**

---

Sintaxe	DbGoBotton()
Descrição	Move o cursor da área de trabalho ativa para o último registro lógico.

## **DBRLOCKLIST()**

---

Sintaxe	DBRLOCKLIST()
Descrição	Retorna um array contendo o record number (recno) de todos os registros, travados da área de trabalho ativa.

## **DBSEEK() E MSSEEK()**

---

Sintaxe	
Descrição	

## DBSKIP()

DbSeek(cChave, lSoftSeek, lLast)

DbSeek: Permite posicionar o cursor da área de trabalho ativo no registro com as informações especificadas na chave de busca, fornecendo um retorno lógico indicando se o posicionamento foi efetuado com sucesso, ou seja, se a informação especificada na chave de busca foi localizada na área de trabalho.

MsSeek(): Função desenvolvida pela área de Tecnologia da Microsiga, a qual possui as mesmas funcionalidades básicas da função DbSeek(), com a vantagem de não necessitar acessar novamente a base de dados para localizar uma informação já utilizada pela thread (conexão) ativa.

Sintaxe	DbSkip(nRegistros)
Descrição	Move o cursor do registro posicionado para o próximo (ou anterior, dependendo do parâmetro), em função da ordem ativa para a área de trabalho.

#### DBSELECTAREA()

Sintaxe	DbSelectArea(nArea   cArea)
Descrição	Define a área de trabalho especificada como sendo a área ativa. Todas as operações subsequentes que fizerem referência a uma área de trabalho para utilização, a menos que a área desejada seja informada explicitamente.

## **DBSETFILTER()**

---

Sintaxe	DbSetFilter(bCondicao, cCondicao)
Descrição	Define um filtro para a área de trabalho ativa, o qual pode ser descrito na forma de um bloco de código ou através de uma expressão simples.

## **DBSETORDER()**

---

Sintaxe	DbSetOrder(nOrdem)
Descrição	Define qual índice será utilizado pela área de trabalho ativa, ou seja, pela área previamente selecionada através do comando DbSelectArea(). As ordens disponíveis no Ambiente Protheus são aquelas definidas no SINDEX /SIX, ou as ordens disponibilizadas por meio de índices temporários.

## **DBORDERNICKNAME()**

---

Sintaxe	DbOrderNickName(NickName)
Descrição	Define qual índice criado pelo usuário será utilizado. O usuário pode incluir os seus próprios índices e no momento da inclusão deve criar o NICKNAME para o mesmo.

## **DBUNLOCK()**

---

Sintaxe	DBUNLOCK()
Descrição	Mesma funcionalidade da função UNLOCK(), só que recomendada para ambientes de rede nos quais os arquivos são compartilhados.  Libera o travamento do registro posicionado na área de trabalho ativa e confirma as atualizações efetuadas naquele registro.

Sintaxe	<b>DBUNLOCKALL()</b>
Descrição	Libera o travamento de todos os registros de todas as áreas de trabalho disponíveis na thread (conexão) ativa.

#### DBUSEAREA()

---

Sintaxe	<b>DbUseArea(lNovo, cDriver, cArquivo, cAlias, lComparilhado,; lSoLeitura)</b>
Descrição	Define um arquivo de base de dados como uma área de trabalho disponível na aplicação.

#### MSUNLOCK()

---

Sintaxe	<b>MsUnLock()</b>
Descrição	Libera o travamento (lock) do registro posicionado, confirmando as atualizações efetuadas neste registro.

#### RECLOCK()

---

Sintaxe	<b>RecLock(cAlias,lInclui)</b>
Descrição	Efetua o travamento do registro posicionado na área de trabalho ativa, permitindo a inclusão ou alteração das informações do mesmo.

#### RLOCK()

Sintaxe	<b>RLOCK() lSucesso</b>
Descrição	Efetua o travamento do registro posicionado na área de trabalho ativa.

## SELECT()

Sintaxe	Select(cArea)
Descrição	Determina o número de referência de um determinado alias em um ambiente de trabalho. Caso o alias especificado não esteja em uso no Ambiente, será retornado o valor 0 (zero).

## SOFTLOCK()

Sintaxe	SoftLock(cAlias)
Descrição	<p>Permite a reserva do registro posicionado na área de trabalho ativa de forma que outras operações, com exceção da atual, não possam atualizar este registro. Difere da função RecLock() pois não gera uma obrigação de atualização, e pode ser sucedido por ele.</p> <p>Na aplicação ERP Protheus, o SoftLock() é utilizado nos browses, antes da confirmação da operação de alteração e exclusão, pois neste momento a mesma ainda não foi efetivada, mas outras conexões não podem acessar aquele registro pois o mesmo está em manutenção, o que implementa da integridade da informação.</p>

## UNLOCK()

Sintaxe	UNLOCK()
Descrição	Libera o travamento do registro posicionado na área de trabalho ativa e confirma as atualizações efetuadas naquele registro.

### **11.3. Diferenciação entre variáveis e os nomes de campos**

Muitas vezes uma variável pode ter o mesmo nome que um campo de um arquivo ou de uma tabela aberta no momento. Neste caso, o ADVPL privilegiará o campo, de forma que uma referência a um nome, que identifique tanto uma variável como um campo, resultará no conteúdo do campo.

Para especificar qual deve ser o elemento referenciado, deve-se utilizar o operador de identificação de apelido (->) e um dos dois identificadores de referência, MEMVAR ou FIELD.

```
cRes := MEMVAR->NOME
```

Essa linha de comando identifica que o valor atribuído à variável cRes deve ser o valor da variável de memória chamada NOME.

```
cRes := FIELD->NOME
```

Nesse caso, o valor atribuído à variável cRes será o valor do campo NOME, existente no arquivo, ou tabela aberto na área atual.

O identificador FIELD pode ser substituído pelo apelido de um arquivo ou tabela abertos, para evitar a necessidade de selecionar a área antes de acessar o conteúdo de determinado campo.

```
cRes := CLIENTES->NOME
```

As tabelas de dados, utilizadas pela aplicação ERP, recebem automaticamente do Sistema o apelido ou ALIAS, especificado para as mesmas no arquivo de sistema SX2. Assim, se o campo NOME pertence a uma tabela da aplicação PROTHEUS, o mesmo poderá ser referenciado com a indicação do ALIAS pré-definido dessa tabela.

```
cRes := SA1->NOME // SA1 – Cadastro de Clientes
```



*Dica*

Para maiores detalhes sobre abertura de arquivos com a atribuição de apelidos, consulte a documentação sobre o acesso ao banco de dados, ou à documentação da função dbUseArea().

Os alias das tabelas da aplicação ERP são padronizados em três letras, que correspondem às iniciais da tabela. As configurações de cada ALIAS, utilizadas pelo Sistema podem ser visualizadas através do módulo Configurador -> Bases de Dados -> Dicionários -> Bases de Dados.

## **11.4. Controle de numeração sequencial**

Alguns campos de numeração do Protheus são fornecidos pelo Sistema em ordem ascendente. É o caso, por exemplo, do número do pedido de venda e outros que servem como identificador das informações das tabelas. É preciso ter um controle do fornecimento desses números, em especial quando vários usuários estão trabalhando simultaneamente.



*Fique atento*

Os campos que recebem o tratamento de numeração sequencial pela aplicação ERP não devem ser considerados como chave primária das tabelas à qual estão vinculados.

No caso específico da aplicação ERP Protheus, a chave primária em Ambientes TOPCONNECT será o campo R\_E\_C\_N\_O\_, e para bases de dados padrão ISAM, o conceito de chave primária é implementado pela regra de negócio do sistema, pois este padrão de dados não possui o conceito de unicidade de dados.

### Semáforos

Para definir o conceito do que é um semáforo de numeração deve-se avaliar a seguinte sequência de eventos no sistema:

Ao ser fornecido um número, ele permanece reservado até a conclusão da operação que o solicitou;

Se esta operação for confirmada o número é indisponibilizado, mas se a operação for cancelada, o número voltará a estar disponível mesmo que naquele momento números maiores já tenham sido oferecidos e utilizados.

Com isso, mesmo que tenhamos vários processos solicitando as numerações sequenciais para uma mesma tabela, como por exemplo inclusões simultâneas de pedidos de vendas, teremos para cada pedido um número exclusivo e sem o intervalos e numerações não utilizadas.

## **Funções de controle de semáforos e numeração sequencial**

A linguagem ADVPL permite a utilização das seguintes funções para o controle das numerações sequenciais utilizadas nas tabelas da aplicação ERP:

GETSXENUM()

CONFIRMSXE()

ROLLBACKSXE()

GETSXENUM()

---

Sintaxe	GETSXENUM(cAlias, cCampo, cAliasSXE, nOrdem)
Descrição	Obtém o número sequência do alias especificado no parâmetro através da referência aos arquivos de sistema SXE/SXF, ou ao servidor de numeração, quando esta configuração está habilitada no Ambiente Protheus.

CONFIRMSXE()

---

Sintaxe	CONFIRMSXE(lVerifica)
Descrição	Confirma o número alocado através do último comando GETSXENUM().

ROLLBACKSXE()

---

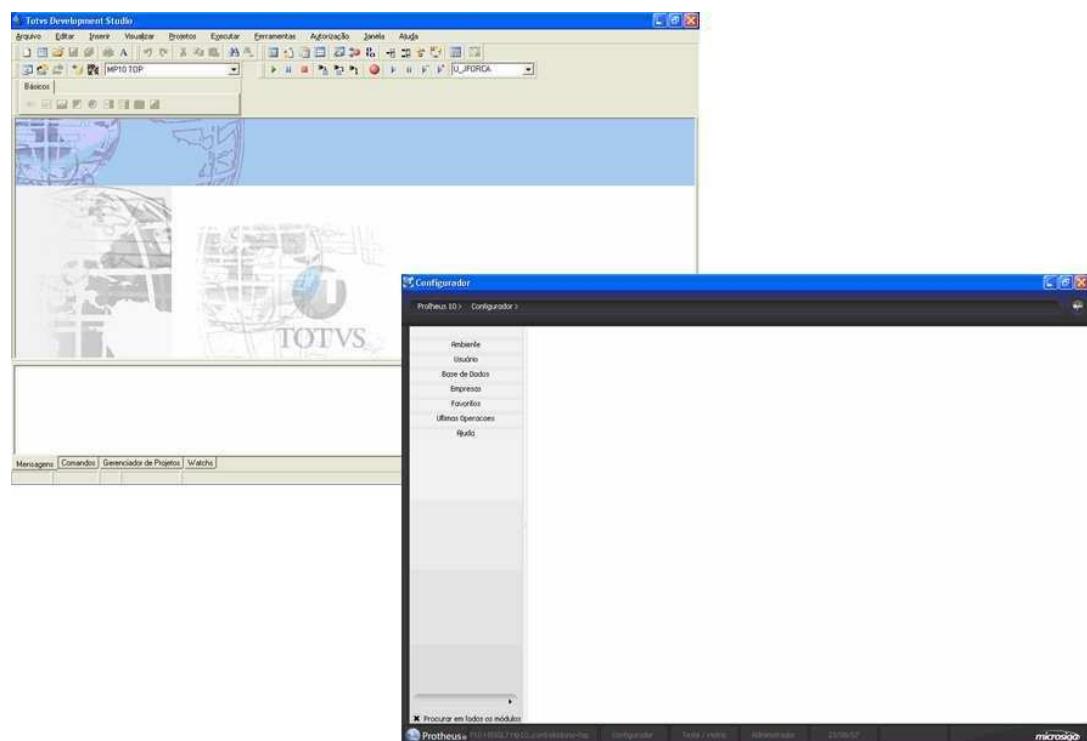
Sintaxe	ROLLBACKSXE()
Descrição	Descarta o número fornecido pelo último comando GETSXENUM(), retornando a numeração disponível para as outras conexões.

## **12. CUSTOMIZAÇÕES PARA A APLICAÇÃO ERP**

Nesse tópico serão abordadas as formas pelas quais a aplicação ERP Protheus pode ser customizada, com a utilização da linguagem ADVPL.

Pelos recursos de configuração da aplicação ERP, disponíveis no módulo Configurador, é possível implementar as seguintes customizações:

- validações de campos e perguntas do Sistema e de usuários;
- inclusão de gatilhos em campos de Sistemas e de usuários;
- inclusão de regras em parâmetros de Sistemas e de usuários;
- Desenvolvimento de pontos de entrada para interagir com funções padrões do Sistema.



## **12.1. Customização de campos – Dicionário de Dados**

## 12.1.1.

### **Validações de campos e perguntas**

As funções de validação têm como característica fundamental um retorno do tipo lógico, ou seja, um conteúdo .T. – Verdadeiro, ou .F. – Falso.

Com base nessa premissa, a utilização de validações, no Dicionário de Dados (SX3), ou nas Perguntas de Processos e Relatórios (SX1), deverá estar focada sempre na utilização de funções, ou expressões que resultem em um retorno lógico.

Com a utilização do módulo Configurador é possível alterar as propriedades de um campo, ou de uma pergunta de forma a incluir as regras de validação para as seguintes situações:

SX3 – validação de usuário (X3\_VLDUSER);

SX1 – validação da pergunta (X1\_VALID).

Dentre as funções que a linguagem ADVPL, em conjunto com os recursos desenvolvidos pela aplicação ERP, para validação de campos e perguntas serão detalhadas:

VAZIO()

NAOVAZIO()

EXISTCPO()

EXISTCHAV()

PERTENCE()

POSITIVO()

NEGATIVO()

TEXTO()

---

EXISTCHAV()

Sintaxe	ExistChav(cAlias, cConteudo, nIndice)
Descrição	<p>Retorna .T. ,ou .F. se o conteúdo especificado existe no alias especificado. Se existir, será exibido um help de Sistema com um aviso informando a ocorrência.</p> <p>Esta função é utilizada normalmente para verificar se um determinado código de cadastro já existe na tabela na qual a informação será inserida, como por exemplo, o CNPJ no cadastro de clientes, ou fornecedores.</p>

## EXISTCPO()

Sintaxe	ExistCpo(cAlias, cConteudo, nIndice)
Descrição	<p>Retorna .T., ou .F. se o conteúdo especificado não existir no alias especificado. Se não existir, será exibido um help de Sistema com um aviso informando a ocorrência.</p> <p>Essa Função é utilizada normalmente para verificar se a informação digitada em um campo, que é dependente de outra tabela, realmente existe na mesma. Como por exemplo, o código de um cliente em um pedido de venda.</p>

## NAOVAZIO()

Sintaxe	NaoVazio()
Descrição	Retorna .T. ou .F. se o conteúdo do campo posicionado no momento não está vazio.

## NEGATIVO()

Sintaxe	Negativo()
Descrição	Retorna .T. ou .F. se o conteúdo digitado para o campo é negativo.

## PERTENCE()

Sintaxe	Pertence(cString)
Descrição	Retorna .T. ou .F. se o conteúdo digitado para o campo está contido na string, definida como o parâmetro da função. Normalmente utilizada em campos com a opção de combo, pois caso contrário seria utilizada a função ExistCpo().

## POSITIVO()

Sintaxe	Positivo()
Descrição	Retorna .T. ou .F. se o conteúdo digitado para o campo é positivo.

## TEXTO()

Sintaxe	Texto()
Descrição	Retorna .T. ou .F. se o conteúdo digitado para o campo contém apenas números, ou alfanuméricos.

## VAZIO()

Sintaxe	Vazio()
Descrição	Retorna .T. ou .F. se o conteúdo do campo posicionado no momento está vazio.

## **12.1.2.**

### **Pictures de formação disponíveis**

Com base na documentação disponível no TDN, a linguagem ADVPL e a aplicação ERP Protheus admitem as seguintes pictures:

---

#### Dicionário de Dados (SX3) e GET

Funções	
Conteúdo	Funcionalidade
A	Permite apenas caracteres alfabéticos
C	Exibe CR depois de números positivos
E	Exibe numérico com o ponto e vírgula invertidos (formato Europeu)
R	Insere caracteres diferentes dos caracteres de template na exibição, mas não os insere na variável do GET
S<n>	Permite o rolamento horizontal do texto dentro do GET, <n>. É um número inteiro que identifica o tamanho da região
X	Exibe DB depois de números negativos
Z	Exibe zeros como brancos
(	Exibe números negativos entre parênteses com os espaços em branco iniciais
)	Exibe números negativos entre parênteses sem os espaços em branco iniciais
!	Converte caracteres alfabéticos para maiúsculos

Templates	
Conteúdo	Funcionalidade
X	Permite qualquer caractere
9	Permite apenas dígitos para qualquer tipo de dado, incluindo o sinal para numéricos

#	Permite dígitos, sinais e espaços em branco para qualquer tipo de dado
!	Converte caracteres alfabéticos para maiúsculo
*	Exibe um asterisco no lugar dos espaços em branco iniciais em números
.	Exibe o ponto decimal
,	Exibe a posição do milhar

Exemplo 01 – Picture campo numérico

CT2\_VALOR – Numérico – 17,2

Picture: @E 99,999,999,999,999.99

Exemplo 02 – Picture campo texto, com digitação apenas em caixa alta

A1\_NOME – Caracter - 40

Picture: @!

## SAY e PSAY

---

Funções	
Conteúdo	Funcionalidade
C	Exibe CR depois de números positivos
E	Exibe numérico com o ponto e a vírgula invertidos (formato Europeu)
R	Insere caracteres diferentes dos caracteres de template
X	Exibe DB depois de números negativos
Z	Exibe zeros como brancos
(	Envolve números negativos entre parênteses
!	Converte todos os caracteres alfabéticos para maiúsculo

Templates	
Conteúdo	Funcionalidade
X	Exibe dígitos para qualquer tipo de dado
9	Exibe dígitos para qualquer tipo de dado
#	Exibe dígitos para qualquer tipo de dado
!	Converte caracteres alfabéticos para maiúsculo
*	Exibe asterisco no lugar de espaços em branco e iniciais em números
.	Exibe a posição do ponto decimal
,	Exibe a posição do milhar

Exemplo 01 – Picture campo numérico

CT2\_VALOR – Numérico – 17,2

Picture: @E 99,999,999,999,999.99



Anotações

---

---

---

---

## **12.2. Customização de gatilhos – Configurador**

A aplicação ERP utiliza o recurso de gatilhos em campo com a finalidade de auxiliar o usuário no preenchimento de informações, durante a sua digitação.. As funções que podem ser utilizadas no gatilho estão diretamente relacionadas à definição da expressão de retorno, que será executada na avaliação do gatilho do campo.

As regras que devem ser observadas na montagem de um gatilho e configuração de seu retorno são:

Na definição da chave de busca do gatilho deve ser avaliada qual filial deverá ser utilizada como parte da chave: a filial da tabela de origem do gatilho, ou a filial da tabela que será consultada. O que normalmente determina a filial que será utilizada, como parte da chave, é justamente a informação que será consultada, aonde:

- As Consultas de informações, entre tabelas com estrutura de cabeçalho e itens, devem utilizar a filial da tabela de origem, pois ambas as tabelas devem possuir o mesmo tratamento de filial (compartilhado ou exclusivo).

**Exemplos:**

Pedido de vendas -> SC5 x SC6  
Nota fiscal de entrada -> SF1 x SD1  
Ficha de imobilizado -> SN1 x SN3  
Orçamento contábil -> CV1 x CV2

- Consulta de informações de tabelas de cadastros devem utilizar a filial da tabela a ser consultada, pois o compartilhamento dos cadastros normalmente é independente em relação às movimentações e outros cadastros do Sistema.

**Exemplos:**

Cadastro de clientes -> SA1 (compartilhado)  
Cadastro de fornecedores -> SA2 (compartilhado)  
Cadastro de vendedores -> SA3 (exclusivo)  
Cadastro de transportadoras -> SA4 (exclusivo)

- Consulta as informações de tabelas de movimentos que devem utilizar a filial da tabela a ser consultada, pois apesar das movimentações de um módulo seguirem um determinado padrão. A consulta pode ser realizada entre tabelas de módulos distintos, o que poderia gerar um retorno incorreto baseado nas diferentes parametrizações desses Ambientes.

**Exemplos:**

Contas a pagar -> SE2 (compartilhado)  
Movimentos contábeis -> CT2 (exclusivo)  
Pedidos de compras -> SC7 (compartilhado)  
Itens da nota fiscal de entrada -> SD1 (exclusivo)

Na definição da regra de retorno deve ser considerado o tipo do campo que será atualizado, pois é esse campo que determina qual o tipo do retorno será considerado válido para o gatilho.

### **12.3. Customização de parâmetros – Configurador**

Os parâmetros de sistema utilizados pela aplicação ERP e definidos através do módulo configurador possuem as seguintes características fundamentais:

Tipo de parâmetro: de forma similar a uma variável, um parâmetro terá um tipo de conteúdo pré-definido em seu cadastro.

Essa informação é utilizada pelas funções da aplicação ERP, na interpretação do conteúdo do parâmetro e no retorno dessa informação à rotina que o consultou.

Interpretação do conteúdo do parâmetro: Diversos parâmetros do Sistema têm seu conteúdo macro executado durante a execução de uma rotina do ERP. Esses parâmetros macro executáveis tem como única característica em comum o seu tipo: caractere, mas não existe nenhum identificador explícito que permite a fácil visualização de quais parâmetros possuem um retorno simples e de quais parâmetros terão o seu conteúdo macro executado para determinar o retorno “real”.

A única forma eficaz de avaliar como um parâmetro é tratado (simples retorno, ou macro execução) é através do código fonte da rotina, que deverá ser avaliado como é tratado o retorno de uma destas funções:

- GETMV()
- SUPERGETMV()
- GETNEWPAR()

Um retorno macro executado é determinado através do uso do operador “&” ou de uma das funções de execução de blocos de código, em conjunto com uma das funções citadas anteriormente.

### **12.3.1.**

### **Funções para a manipulação de parâmetros**

A aplicação ERP disponibiliza as seguintes funções para consulta e atualização de parâmetros:

GETMV()

SUPERGETMV()

GETNEWPAR()

PUTMV()

GETMV()

---

Sintaxe	GETMV(cParametro)
Descrição	Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Se o parâmetro não existir, será exibido um help do sistema informando a ocorrência.

## GETNEWPAR()

Sintaxe	GETNEWPAR(cParametro, cPadrao, cFilial)
Descrição	<p>Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Se o parâmetro não existir, será exibido um help do Sistema informando a ocorrência.</p> <p>Difere do SuperGetMV() pois considera que o parâmetro pode não existir na versão atual do Sistema, e por consequência não será exibida a mensagem de help.</p>

## PUTMV()

Sintaxe	PUTMV(cParametro, cConteudo)
Descrição	Atualiza o conteúdo do parâmetro especificado no arquivo SX6, de acordo com as parametrizações informadas.

## SUPERGETMV()

Sintaxe	SUPERGETMV(cParametro , lHelp , cPadrao , cFilial)
Descrição	<p>Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Se o parâmetro não existir, será exibido um help do Sistema informando a ocorrência.</p> <p>Difere do GetMv() pois os parâmetros consultados são adicionados em uma área de memória que permite que em uma nova consulta não seja necessário acessar e pesquisar o parâmetro na base de dados.</p>

### **12.3.2.**

### **Cuidados na utilização de um parâmetro**

Um parâmetro de sistema tem a finalidade de propiciar um retorno válido a um conteúdo previamente definido na configuração do módulo para uma rotina, processo, ou quaisquer outros tipos de funcionalidades disponíveis na aplicação.

Apesar de haver parâmetros que permitem a configuração de expressões, e por consequência a utilização de funções para definir o retorno que será obtido com a consulta desse parâmetro, é expressamente proibido o uso de funções em parâmetros para manipular as informações da base de dados do Sistema.

Se houver necessidade de ser implementado um tratamento adicional a um processo padrão do Sistema, o mesmo deverá utilizar o recurso de ponto de entrada.

A razão dessa restrição é simples:

As rotinas da aplicação ERP não protegem a consulta de conteúdos de parâmetros, quanto às gravações realizadas dentro, ou fora de uma transação.

Dessa forma, qualquer alteração na base realizada por uma rotina, configurada em um parâmetro, pode ocasionar a perda da integridade das informações do Sistema.

## **12.4. Pontos de Entrada – Conceitos, Premissas e Regras**

### **Conceitos**

---

Um ponto de entrada é uma User Function desenvolvida com a finalidade de interagir com uma rotina padrão da aplicação ERP.

A User Function deverá ter um nome pré-estabelecido no desenvolvimento da rotina padrão do ERP, e de acordo com esta pré-disposição e o momento no qual o ponto de entrada é executado durante um processamento, ele poderá:

Complementar uma validação realizada pela aplicação;

Complementar as atualizações realizadas pelo processamento em tabelas padrões do ERP;

Implementar a atualização de tabelas específicas durante o processamento de uma rotina padrão do ERP;

Executar uma ação sem processos de atualizações, mas que necessite utilizar as informações atuais do Ambiente, durante o processamento da rotina padrão para determinar as características do processo;

Substituir um processamento padrão do Sistema por uma regra específica do cliente, no qual o mesmo será implementado.

### **Premissas e Regras**

---

Um ponto de entrada não deve ser utilizado para outras finalidades senão para as quais foi pré-definido, sob pena de causar a perda da integridade das informações da base de dados, ou provocar eventos de erro durante a execução da rotina padrão.

Um ponto de entrada deve ser transparente para o processo padrão, de forma que todas as tabelas, acessadas pelo ponto de entrada e que sejam utilizadas pela rotina padrão, deverão ter a sua situação imediatamente anterior à execução do ponto

restaurado ao término do mesmo, e para isto recomenda-se o uso das funções GETAREA() e RESTAREA().

Como um ponto de entrada não é executado da forma tradicional, ou seja, ele não é chamado como uma função, ele não recebe parâmetros. A aplicação ERP disponibiliza uma variável de Sistema denominada PARAMIXB, a qual recebe os parâmetros da função chamadora e os disponibiliza para serem utilizados pela rotina customizada.

A variável PARAMIXB não possui um padrão de definição nos códigos fontes da aplicação ERP, desta forma seu tipo pode variar de um conteúdo simples (caractere, numérico, lógico e etc.) a um tipo complexo como um array ou um objeto. Assim, é necessário sempre avaliar a documentação sobre o ponto, bem como proteger a função customizada de tipos de PARAMIXB não tratados por ela.

## **13. INTERFACES VISUAIS**

A linguagem ADVPL possui duas formas distintas para definição de interfaces visuais no Ambiente ERP: sintaxe convencional, nos padrões da linguagem CLIPPER e a sintaxe orientada a objetos.

Além das diferentes sintaxes disponíveis para definição das interfaces visuais, o ERP Protheus possui funcionalidades pré-definidas, as quais já contêm todos os tratamentos necessários para atender as necessidades básicas de acesso e manutenção das informações do Sistema.

Neste tópico serão abordadas as sintaxes convencionais para definição das interfaces visuais da linguagem ADVPL e as interfaces de manutenção, disponíveis no Ambiente ERP Protheus.

### **13.1. Sintaxe e componentes das interfaces visuais**

A sintaxe convencional para definição de componentes visuais da linguagem ADVPL depende diretamente, em include especificado no cabeçalho, do fonte. Os dois includes, disponíveis para o Ambiente ADVPL Protheus, são:

RWMAKE.CH: Permite a utilização da sintaxe CLIPPER na definição dos componentes visuais.

PROTHEUS.CH: Permite a utilização da sintaxe ADVPL convencional, a qual é um aprimoramento da sintaxe CLIPPER, com a inclusão de novos atributos para os componentes visuais disponibilizados no ERP Protheus.

Para ilustrar a diferença na utilização destes dois includes, seguem as diferentes definições para o componentes Dialog e MsDialog:

Exemplo 01 – Include Rwmake.ch

```
#include "rwmakch.ch"  
@ 0,0 TO 400,600 DIALOG oDlg TITLE "Janela em sintaxe Clipper"  
ACTIVATE DIALOG oDlg CENTERED
```

Exemplo 02 – Include Protheus.ch

```
#include "protheus.ch"  
  
DEFINE MSDIALOG oDlg TITLE "Janela em sintaxe ADVPL "FROM 000,000 TO 400,600  
PIXEL  
ACTIVATE MSDIALOG oDlg CENTERED
```



*Fique atento*

Ambas sintaxes produzirão o mesmo efeito quando compiladas e executadas no ambiente Protheus, mas deve ser utilizada sempre a sintaxe ADVPL, por meio do uso do include PROTHEUS.CH.

Os componentes da interface visual que serão tratados neste tópico, utilizando a sintaxe ADVPL são:

MSDIALOG()

MSGET()

SAY()

BUTTON()

SBUTTON()

### BUTTON()

Sintaxe	@ nLinha,nColuna BUTTON cTexto SIZE nLargura,nAltura UNIDADE OF oObjetoRef ACTION AÇÃO
Descrição	Define o componente visual Button, o qual permite a inclusão de botões de operação na tela da interface, os quais serão visualizados somente com um texto simples para sua identificação.

### MSDIALOG()

Sintaxe	DEFINE MSDIALOG oObjetoDLG TITLE cTitulo FROM nLinIni,nColIni TO nLiFim,nColFim OF oObjetoRef UNIDADE
Descrição	Define o componente MSDIALOG(), que é utilizado como base para os demais componentes da interface visual, pois um componente MSDIALOG() é uma janela da aplicação.

## MSGGET()

Sintaxe	@ nLinha, nColuna MSGET VARIABEL SIZE nLargura,nAltura UNIDADE OF oObjetoRef F3 cF3 VALID VALID WHEN WHEN PICTURE cPicture
Descrição	Define o componente visual MSGET, que é utilizado para captura de informações digitáveis na tela da interface.

## SAY()

Sintaxe	@ nLinha, nColuna SAY cTexto SIZE nLargura,nAltura UNIDADE OF oObjetoRef
Descrição	Define o componente visual SAY, que é utilizado para a exibição de textos em uma tela de interface.

## SBUTTON()

Sintaxe	DEFINE SBUTTON FROM nLinha, nColuna TYPE N ACTION AÇÃO STATUS OF oObjetoRet
Descrição	Define o componente visual SButton, que permite a inclusão de botões de operação na tela da interface, que serão visualizados dependendo da interface do sistema ERP, utilizada somente com um texto simples para sua identificação, ou com uma imagem (BitMap) pré-definido.

## Interface visual completa

Abaixo, segue um código completo de interface, utilizando todos os elementos da interface visual descritos anteriormente:

```
DEFINE MSDIALOG oDlg TITLE cTitulo FROM 000,000 TO 080,300 PIXEL
@ 001,001 TO 040, 150 OF oDlg PIXEL
@ 010,010 SAY cTexto SIZE 55, 07 OF oDlg PIXEL
```

```
@ 010,050 MSGET cCGC      SIZE 55, 11 OF oDlg PIXEL PICTURE "@R
99.999.999/9999-99";
VALID !Vazio()

DEFINE SBUTTON FROM 010, 120 TYPE 1 ACTION (nOpca := 1,oDlg:End());
ENABLE OF oDlg

DEFINE SBUTTON FROM 020, 120 TYPE 2 ACTION (nOpca := 2,oDlg:End());
ENABLE OF oDlg

ACTIVATE MSDIALOG oDlg CENTERED
```



*Dica*

O código demonstrado anteriormente é utilizado nos exercícios de fixação deste material e deverá produzir a seguinte interface:



## **13.2. Interfaces padrões para atualizações de dados**

Os programas de atualização de cadastros e digitação de movimentos seguem um padrão que se apóia no Dicionário de Dados.

Basicamente são duas as interfaces quer permitem a visualização das informações e a manipulação dos dados do Sistema.

AxCadastro  
Mbrowse

Ambos modelos utilizam como premissa que a estrutura da tabela a ser utilizada esteja definida no dicionário de dados do sistema (SX3).

O AxCadastro() é uma funcionalidade de cadastro simples, com poucas opções de customização, composta por:

Browse padrão para visualização das informações da base de dados, de acordo com as configurações do SX3 – Dicionário de Dados (campo browse).

Funções de pesquisa, visualização, inclusão, alteração e exclusão de padrões para visualização de registros simples, sem a opção de cabeçalho e itens.

Sintaxe: AxCadastro(cAlias, cTitulo, cVldExc, cVldAlt)

Parâmetros:

cAlias	Alias padrão do sistema para utilização, o qual deve estar definido no dicionário de dados – SX3
cTitulo	Título da Janela
cVldExc	Validação para Exclusão
cVldAlt	Validação para Alteração

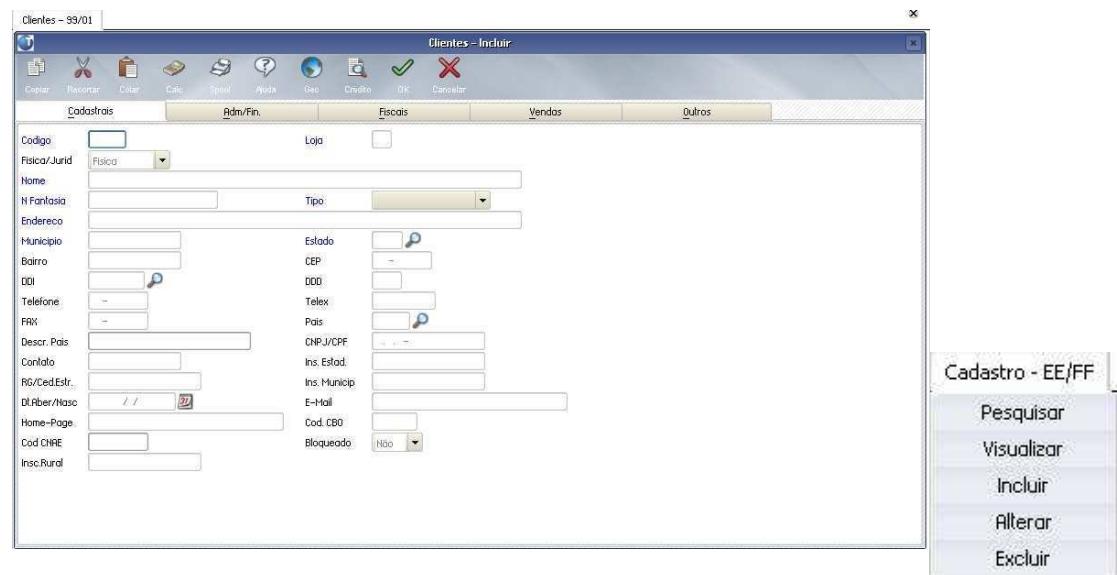
Exemplo:

```
#include "protheus.ch"
```

```
User Function XCadSA2()
```

```
Local cAlias := "SA2"  
Local cTitulo := "Cadastro de Fornecedores"  
Local cVldExc := ".T."  
Local cVldAlt := ".T."
```

```
dbSelectArea(cAlias)  
dbSetOrder(1)  
AxCadastro(cAlias,cTitulo,cVldExc,cVldAlt)  
Return
```



### **13.2.2.**

### **MBrowse()**

A Mbrowse() é uma funcionalidade de cadastro que permite a utilização de recursos mais aprimorados na visualização e manipulação das informações do Sistema, possuindo os seguintes componentes:

Browse padrão para visualização das informações da base de dados, de acordo com as configurações do SX3 – Dicionário de Dados (campo browse).

Parametrização para as funções específicas para as ações de visualização, inclusão, alteração e exclusão de informações, o que viabiliza a manutenção de informações com estrutura de cabeçalhos e itens.

Recursos adicionais como identificadores de status de registros, legendas e filtros para as informações.

Sintaxe simplificada: MBrowse(nLin1, nCol1, nLin2, nCol2, cAlias)

#### **Parâmetros**

nLin1, nCol1, nLin2, nCol2	Coordenadas dos cantos aonde o browse será exibido. Para seguir o padrão da AXCADASTRO() use 6,1,22,75 .
cAlias	Alias padrão do sistema para utilização. Deve estar definido no dicionário de dados – SX3.

## Variáveis private adicionais

	<p>Array contendo as funções que serão executadas pela Mbrowse. Este array pode ser parametrizado com as funções básicas da AxCadastro conforme abaixo:</p> <pre> AADD(aRotina,{"Pesquisar" , "AxPesqui",0,1}) AADD(aRotina,{"Visualizar" , "AxVisual",0,2}) AADD(aRotina,{"Incluir" , "AxInclui",0,3}) AADD(aRotina,{"Alterar" , "AxAltera",0,4}) AADD(aRotina,{"Excluir" , "AxDelete",0,5}) </pre>
cCadastro	Título do browse que será exibido.

Exemplo:

```

#include "protheus.ch"

User Function MBrwSA2()

Local cAlias := "SA2"
Private cCadastro := "Cadastro de Fornecedores"
Private aRotina      := {}

AADD(aRotina,{ "Pesquisar" , "AxPesqui",0,1})
AADD(aRotina,{ "Visualizar" , "AxVisual",0,2})
AADD(aRotina,{ "Incluir" , "AxInclui",0,3})
AADD(aRotina,{ "Alterar" , "AxAltera",0,4})
AADD(aRotina,{ "Excluir" , "AxDelete",0,5})

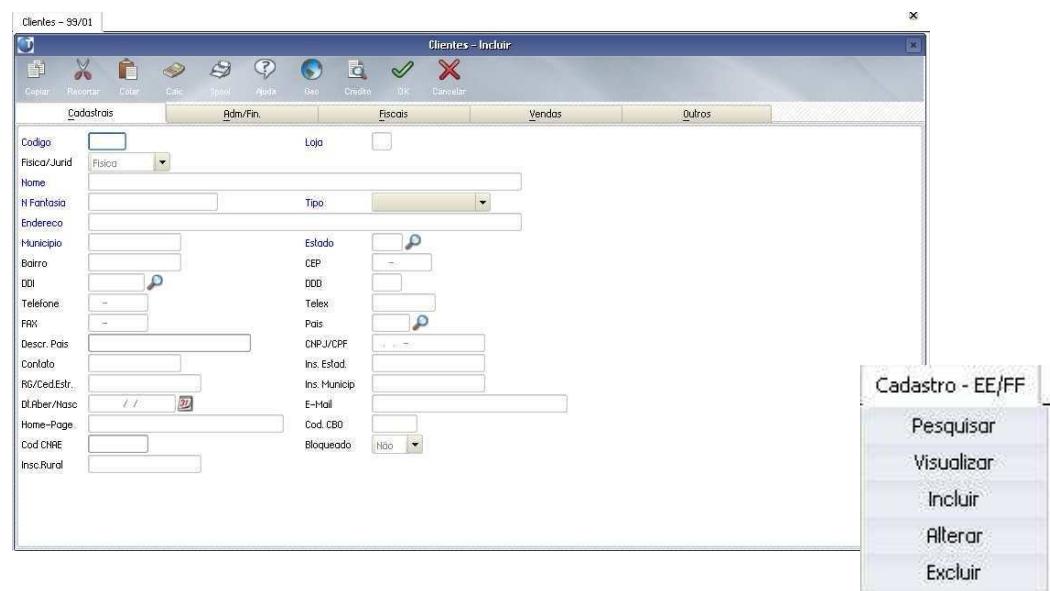
```

```

dbSelectArea(cAlias)
dbSetOrder(1)
mBrowse(6,1,22,75,cAlias)

Return

```



**Fique  
atento**

Utilizando a parametrização exemplificada, o efeito obtido com o uso da Mbrowse() será o mesmo obtido com o uso da AxCadastro().

A posição das funções no array aRotina define o conteúdo de uma variável de controle que será repassada para as funções chamadas a partir da Mbrowse, convencionada como nOpc. Dessa forma, para manter o padrão da aplicação ERP a ordem a ser seguida na definição do aRotina é;

- 1 – Pesquisar;
- 2 – Visualizar;
- 3 – Incluir;

4 – Alterar;

5 – Excluir;

6 – Livre.



*Dica*

Ao definir as funções no array aRotina, se o nome da função não for especificado com “()”, a Mbrowse passará como parâmetros as seguintes variáveis de controle:

cAlias: Alias ativo definido para a Mbrowse;

nRecno: Record number (recno) do registro posicionado no alias ativo;

nOpc: Posição da opção utilizada na Mbrowse de acordo com a ordem da função no array a Rotina.

Exemplo: Função BIclui() substituindo a função AxInclui()

```
#include "protheus.ch"
```

```
User Function MBrwSA2()
```

```
Local cAlias := "SA2"
```

```
Private cCadastro := "Cadastro de Fornecedores"
```

```
Private aRotina      := {}
```

```
AADD(aRotina,{ "Pesquisar"  , "AxPesqui"  ,0,1})
```

```
AADD(aRotina,{ "Visualizar" , "AxVisual"   ,0,2})
```

```
AADD(aRotina,{ "Incluir"    , "U_BInclui"  ,0,3})
```

```
AADD(aRotina,{ "Alterar"    , "AxAltera"   ,0,4})
```

```
AADD(aRotina,{ "Excluir"    , "AxDelete"   ,0,5})
```

```
dbSelectArea(cAlias)
```

```
dbSetOrder(1)
```

```
mBrowse(6,1,22,75,cAlias)
```

```
Return
```

```
USER FUNCTION BIclui(cAlias, nReg, nOpc)
```

```
Local cTudoOk := "(Alert('OK'),.T.)"
```

```
AxInclui(cAlias,nReg,nOpc,,,cTudoOk)
```

```
RETURN
```



Anotações

---

---

---

---

### **13.2.3.**

### **AxFunctions()**

Conforme foi mencionado nos tópicos sobre as interfaces padrões AxCadastro() e Mbrowse(), existem funções padrões da aplicação ERP que permitem a visualização, inclusão, alteração e exclusão de dados em formato simples.

Essas funções são padrões na definição da interface AxCadastro() e podem ser utilizadas também da construção no array aRotina, utilizado pela Mbrowse(), as quais estão listadas a seguir:

AXPESQUI()

AXVISUAL()

AXINCLUI()

AXALTERA()

AXDELETA()

AXALTERA()

---

Sintaxe	AxAltera(cAlias, nReg, nOpc, aAcho, cFunc, aCpos, cTudoOk, lF3,; cTransact, aButtons, aParam, aAuto, lVirtual, lMaximized)
Descrição	Função de alteração padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

AXDELETA()

---

Sintaxe	AXDELETA(cAlias, nReg, nOpc, cTransact, aCpos, aButtons, aParam,; aAuto, lMaximized)
Descrição	Função de exclusão padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

AXINCLUI()

---

Sintaxe	AxInclui(cAlias, nReg, nOpc, aAcho, cFunc, aCpos, cTudoOk, lf3,; cTransact, aButtons, aParam, aAuto, lVirtual, lMaximized)
Descrição	Função de inclusão padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

### AXPESQUI()

---

Sintaxe	AXPESQUI()
Descrição	Função de pesquisa padrão em registros exibidos pelos browses do Sistema, que posiciona o browse no registro pesquisado. Exibe uma tela que permite a seleção do índice a ser utilizado na pesquisa e a digitação das informações que compõe a chave de busca.

## AXVISUAL()

---

Sintaxe	AXVISUAL(cAlias, nReg, nOpc, aAcho, nColMens, cMensagem, cFunc,; aButtons, lMaximized )
Descrição	Função de visualização padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().



### Exercícios

#### Exercício 15

Desenvolver uma validação para um campo específico do tipo caractere, cujo conteúdo esteja relacionado a outra tabela e que exiba uma mensagem de aviso caso o código informado não exista nessa tabela relacionada.



### Exercícios

#### Exercício 16

Desenvolver uma validação para um campo caractere existente na base, para que seja avaliado se aquele código já existe cadastrado, e se positivo, exiba uma mensagem de aviso alertando sobre a ocorrência.



### Exercícios

#### Exercício 17

Desenvolver um gatilho que retorne uma descrição complementar para um campo vinculado ao campo código utilizado nos exercícios anteriores.



### Exercícios

#### Exercício 18

Customizar o parâmetro que define o prefixo do título de contas a pagar, gerado pela integração COMPRAS -> FINANCEIRO através da inclusão de uma nota fiscal de entrada, de forma que o prefixo do título seja gerado com o código da filial corrente.



### Exercícios

#### Exercício 19

Proteger a rotina desenvolvida no exercício anterior, de forma a garantir que na utilização da filial como prefixo do título, não ocorrerá duplicidade de dados em contas a pagar do financeiro.



### Exercícios

#### Exercício 20

Vamos Implementar uma validação adicional no cadastro de clientes, com a utilização do ponto de entrada adequado, de forma que o campo CNPJ (A1\_CGC) seja obrigatório para todos os tipos de clientes, exceto os definidos como Exterior.

## APÊNDICES

### BOAS PRÁTICAS DE PROGRAMAÇÃO

#### 14. UTILIZAÇÃO DE IDENTAÇÃO

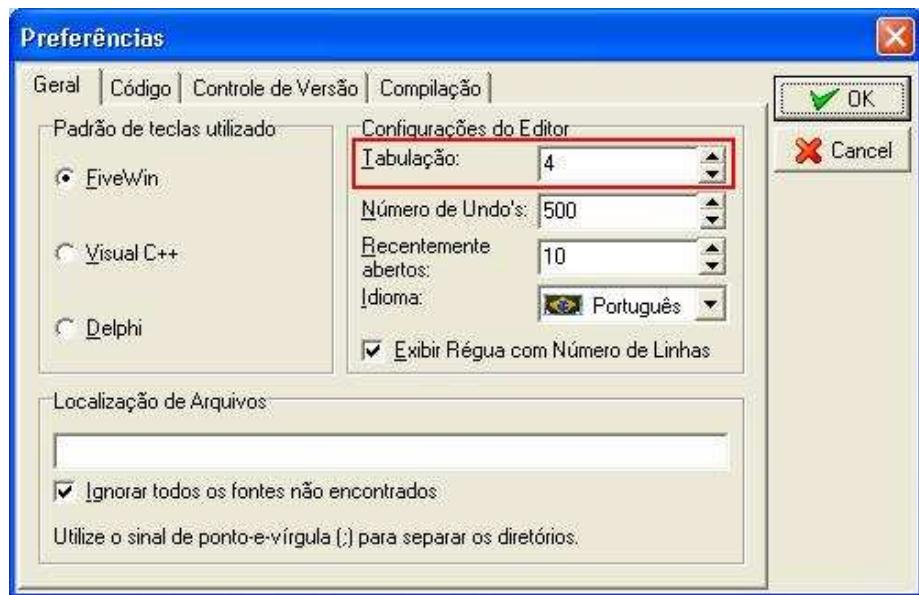
É obrigatória a utilização da identação, pois torna o código muito mais legível. Veja os exemplos abaixo:

```
while !SB1->(EOF)
  If mv_par01 = SB1->B1_COD
    dbSkip()
    Loop
  Endif
  Do Case
    Case SB1->B1_LOCAL == "01" .Or. SB1->B1_LOCAL == "02"
      TrataLocal(SB1->B1_COD,SB1->B1_LOCAL)
    Case SB1->B1_LOCAL == "03"
      TrataDefeito(SB1->B1_COD)
    OtherWise
      TrataCompra(SB1->B1_COD,SB1->B1_LOCAL)
    EndCase
    dbSkip()
  EndDo
```

A utilização da identação, seguindo as estruturas de controle de fluxo (while, if, caso etc), torna a compreensão do código muito mais fácil:

```
If mv_par01 = SB1->B1_COD
  dbSkip()
  Loop
Endif
Do Case
  Case SB1->B1_LOCAL == "01" .Or. SB1->B1_LOCAL == "02"
    TrataLocal(SB1->B1_COD,SB1->B1_LOCAL)
  Case SB1->B1_LOCAL == "03"
    TrataDefeito(SB1->B1_COD)
  OtherWise
    TrataCompra(SB1->B1_COD,SB1->B1_LOCAL)
  EndCase
  dbSkip()
EndDo
```

Para indentar o código utilize a tecla <TAB> e a ferramenta DEV-Studio, a qual pode ser configurada através da opção "Preferências":



## **15. CAPITULAÇÃO DE PALAVRAS-CHAVE**

Uma convenção amplamente utilizada é a de capitular as palavras chaves, funções, variáveis e campos utilizando uma combinação de caracteres em maiúsculo e minúsculo, visando facilitar a leitura do código fonte. O código a seguir:

```
local ncnt while ( ncnt++ < 10 ) ntotal += ncnt * 2 enddo
```

Ficaria melhor com as palavras chaves e variáveis capituladas:

```
Local nCnt While ( nCnt++ < 10 ) nTotal += nCnt * 2 EndDo
```



*Fique  
atento*

Para as funções de manipulação de dados que comecem por “db”, a capitulação só será efetuada após o “db”:

dbSeek()

dbSelectArea()



*Anotações*

---

---

---

---

## **15.1. Palavras em maiúsculo**

A regra é utilizar caracteres em maiúsculo para:

Constantes:

```
#define NUMLINES 60 #define NUMPAGES 1000
```

Variáveis de memória:

```
M-> CT2_CRCNV M->CT2_MCONVER := CriaVar("CT2_CONVER")
```

Campos:

```
SC6->C6_NUMPED
```

Querys:

```
SELECT * FROM...
```

## **16. UTILIZAÇÃO DA NOTAÇÃO HÚNGARA**

A notação húngara consiste em adicionar os prefixos aos nomes de variáveis, de modo a facilmente identificar o seu tipo. Isso facilita a criação de códigos-fonte extensos, pois usando a Notação Húngara, você não precisa ficar o tempo todo voltando à definição de uma variável para se lembrar de qual é o tipo de dados que deve ser colocado nela. Variáveis devem ter um prefixo de Notação Húngara em minúsculas, seguido de um nome que identifique a função da variável, sendo que a inicial de cada palavra deve ser maiúscula.

É obrigatória a utilização dessa notação para nomear variáveis.

Notação	Tipo de dado	Exemplo
a	Array	aValores
b	Bloco de código	bSeek
c	Caracter	cNome
d	Data	dDataBase
l	Lógico	lContinua
n	Numérico	nValor



*Anotações*

---

---

---

---

## **17. PALAVRAS RESERVADAS**

AADD	DTOS	INKEY	REPLICATE	VAL
ABS	ELSE	INT	RLOCK	VALTYPE
ASC	ELSEIF	LASTREC	ROUND	WHILE
AT	EMPTY	LEN	ROW	WORD
BOF	ENDCASE	LOCK	RTRIM	YEAR
BREAK	ENDDO	LOG	SECONDS	CDOW
ENDIF	LOWER	SELECT	CHR	EOF
LTRIM	SETPOS	CMONTH	EXP	MAX
SPACE	COL	FCOUNT	MIN	SQRT
CTOD	FIELDNAME	MONTH	STR	DATE
FILE	PCOL	SUBSTR	DAY	FLOCK
PCOUNT	TIME	DELETED	FOUND	PROCEDURE
TRANSFORM	DEVPOS	FUNCTION	PROW	TRIM
DOW	IF	RECCOUNT	TYPE	DTOC
IIF	RECNO	UPPER	TRY	AS
CATCH	THROW			



*Fique  
atento*

As palavras reservadas não podem ser utilizadas para variáveis, procedimentos ou funções;

As Funções reservadas são pertencentes ao compilador e não podem ser redefinidas por uma aplicação;

Todos os identificadores, que começarem com dois ou mais caracteres “\_”, são utilizados como identificadores internos e são reservados.

Os Identificadores de escopo PRIVATE ou PUBLIC utilizados em aplicações específicas, desenvolvidas por ou para clientes, devem ter sua identificação iniciada por um caractere “\_”.

## **GUIA DE REFERÊNCIA RÁPIDA: Funções e Comandos**

### **ADVPL**

Neste guia de referência rápida serão descritas as funções básicas da linguagem ADVPL, incluindo as funções herdadas da linguagem Clipper, necessárias ao desenvolvimento no Ambiente ERP.

#### **Conversão entre tipos de dados**

##### **CTOD()**

Realiza a conversão de uma informação do tipo caracter no formato “DD/MM/AAAA”, para uma variável do tipo data.

Sintaxe: CTOD(cData)

Parâmetros

cData	Caractere no formato “DD/MM/AAAA”.

Exemplo:

```
cData := "31/12/2006"  
dData := CTOD(cData)  
  
IF dDataBase >= dData  
    MSGALERT("Data do sistema fora da competência")  
ELSE  
    MSGINFO("Data do sistema dentro da competência")  
ENDIF
```

---

CVALTOCHAR()

Realiza a conversão de uma informação do tipo numérico em uma string, sem a adição de espaços a informação.

Sintaxe: CVALTOCHAR(nValor)

Parâmetros

nValor	Valor numérico que será convertido para caractere.
--------	--

Exemplo:

```
FOR nPercorridos := 1 to 10
    MSGINFO("Passos percorridos: "+CvalToChar(nPercorridos))
NEXT nPercorridos
```

## DTOC()

Realiza a conversão de uma informação do tipo data para caractere, sendo o resultado no formato “DD/MM/AAAA”.

Sintaxe: DTOC(dData)

Parâmetros

dData	Variável com conteúdo data.
-------	-----------------------------

Exemplo:

```
MSGINFO("Database do sistema: "+DTOC(dData))
```

## DTOS()

Realiza a conversão de uma informação do tipo data em um caractere, sendo o resultado no formato “AAAAMMDD”.

Sintaxe: DTOS(dData)

Parâmetros

dData	Variável com conteúdo data.
-------	-----------------------------

Exemplo:

```
cQuery := "SELECT A1_COD, A1_LOJA, A1_NREDUZ FROM SA1010 WHERE "
cQuery += "A1_DULTCOM >=""+DTOS(dDataIni)+""
```

## STOD()

Realiza a conversão de uma informação do tipo caractere, com conteúdo no formato “AAAAMMDD” em data.

Sintaxe: STOD(sData)

Parâmetros

sData	String no formato “AAAAMMDD”.
-------	-------------------------------

Exemplo:

```
sData := LERSTR(01,08) // Função que realiza a leitura de uma string de um txt previamente  
                      // aberto  
dData := STOD(sData)
```

## STR()

Realiza a conversão de uma informação do tipo numérico em uma string, adicionando espaços à direita.

Sintaxe: STR(nValor)

Parâmetros

nValor	Valor numérico que será convertido para caractere.
--------	--

Exemplo:

```
FOR nPercorridos := 1 to 10
    MSGINFO("Passos percorridos: "+CvalToChar(nPercorridos))
NEXT nPercorridos
```

### STRZERO()

Realiza a conversão de uma informação do tipo numérico em uma string, adicionando zeros à esquerda do número convertido, de forma que a string gerada tenha o tamanho especificado no parâmetro.

Sintaxe: STRZERO(nValor, nTamanho)

Parâmetros

nValor	Valor numérico que será convertido para caractere.
nTamanho	Tamanho total desejado para a string retornada.

Exemplo:

```
FOR nPercorridos := 1 to 10
    MSGINFO("Passos percorridos: "+CvalToChar(nPercorridos))
NEXT nPercorridos
```

## VAL()

Realiza a conversão de uma informação do tipo caracter em numérica.

Sintaxe: VAL(cValor)

Parâmetros

cValor	String que será convertida para numérico.
--------	---

Exemplo:

```
Static Function Modulo11(cData)
LOCAL L, D, P := 0
L := Len(cdata)
D := 0
P := 1
While L > 0
    P := P + 1
    D := D + (Val(SubStr(cData, L, 1)) * P)
    If P = 9
        P := 1
    End
    L := L - 1
End
D := 11 - (mod(D,11))
If (D == 0 .Or. D == 1 .Or. D == 10 .Or. D == 11)
    D := 1
End
Return(D)
```

## **Verificação de tipos de variáveis**

**TYPE()**

---

Determina o tipo do conteúdo de uma variável, a qual não foi definida na função em execução.

Sintaxe: TYPE("cVariavel")

Parâmetros

"cVariavel"	Nome da variável que se deseja avaliar, entre aspas ("").
-------------	---

Exemplo:

```
IF TYPE("dDataBase") == "D"  
    MSGINFO("Database do sistema: "+DTOC("dDataBase"))  
ELSE  
    MSGINFO("Variável indefinida no momento")
```

## VALTYPE()

Determina o tipo do conteúdo de uma variável, a qual não foi definida na função em execução.

Sintaxe: VALTYPE(cVariavel)

Parâmetros

cVariavel	Nome da variável que se deseja avaliar.
-----------	---

Exemplo:

```
STATIC FUNCTION GETTEXTO(nTamanho, cTitulo, cSay)

LOCAL cTexto := ""
LOCAL nColF      := 0
LOCAL nLargGet   := 0
PRIVATE oDlg

Default cTitulo := "Tela para informar texto"
Default cSay    := "Informe o texto:"
Default nTamanho := 1

IF ValType(nTamanho) != "N" // Se o parâmetro foi passado incorretamente
    nTamanho := 1
ENDIF

cTexto      := Space(nTamanho)
nLargGet    := Round(nTamanho * 2.5,0)
nColF       := Round(195 + (nLargGet * 1.75),0)

DEFINE MSDIALOG oDlg TITLE cTitulo FROM 000,000 TO 120,nColF PIXEL
@ 005,005 TO 060, Round(nColF/2,0) OF oDlg PIXEL
```

```
@ 010,010 SAY cSay SIZE 55, 7 OF oDlg PIXEL
@ 010,065 MSGET cTexto SIZE nLargGet, 11 OF oDlg PIXEL ;
Picture "@!" VALID !Empty(cTexto)
DEFINE SBUTTON FROM 030, 010 TYPE 1 ;
ACTION (nOpca := 1,oDlg:End()) ENABLE OF oDlg
DEFINE SBUTTON FROM 030, 040 TYPE 2 ;
ACTION (nOpca := 0,oDlg:End()) ENABLE OF oDlg
ACTIVATE MSDIALOG oDlg CENTERED

cTexto := IIF(nOpca==1,cTexto,"")

RETURN cTexto
```

## Manipulação de arrays

### Array()

A função Array() é utilizada na definição de variáveis de tipo array, como uma opção a sintaxe utilizando chaves (“{ }”).

Sintaxe: Array(nLinhas, nColunas)

Parâmetros

nLinhas	Determina o número de linhas com as quais o array será criado.
nColunas	Determina o número de colunas com as quais o array será criado.

Exemplo:

```
aDados := Array(3,3) // Cria um array de três linhas, cada qual com 3 colunas.
```



*Fique  
atento*

O array definido pelo comando Array() apesar de já possuir a estrutura solicitada, não possui conteúdo em nenhum de seus elementos, ou seja:

aDados[1] -> array de três posições

aDados[1][1] -> posição válida, mas de conteúdo nulo.

## AADD()

A função AADD() permite a inserção de um item em um array já existente, sendo que este item podem ser um elemento simples, um objeto ou outro array.

Sintaxe: AADD(aArray, xItem)

Parâmetros

aArray	Array pré-existente no qual será adicionado o item definido em xItem.
xItem	Item que será adicionado ao array.

Exemplo:

aDados := {} // Define que a variável aDados é um array, sem especificar suas dimensões.

aItem := {} // Define que a variável aItem é um array, sem especificar suas dimensões.

AADD(aItem, cVariavel1) // Adiciona um elemento no array aItem de acordo com o cVariavel1

AADD(aItem, cVariavel2) // Adiciona um elemento no array aItem de acordo com o cVariavel2

AADD(aItem, cVariavel3) // Adiciona um elemento no array aItem de acordo com o cVariavel3

// Neste ponto o array a Item possui 03 elementos os quais podem ser acessados com:

// aItem[1] -> corresponde ao conteúdo de cVariavel1

// aItem[2] -> corresponde ao conteúdo de cVariavel2

// aItem[3] -> corresponde ao conteúdo de cVariavel3

AADD(aDados,aItem) // Adiciona no array aDados o conteúdo do array aItem

Exemplo (continuação):

// Neste ponto, o array a aDados possui apenas um elemento, que também é um array

// contendo 03 elementos:

// aDados [1][1] -> corresponde ao conteúdo de cVariavel1

// aDados [1][2] -> corresponde ao conteúdo de cVariavel2

// aDados [1][3] -> corresponde ao conteúdo de cVariavel3

AADD(aDados, aItem)

AADD(aDados, aItem)

// Neste ponto, o array aDados possui 03 elementos, aonde cada qual é um array com outros

// 03 elementos, sendo:

// aDados [1][1] -> corresponde ao conteúdo de cVariavel1

// aDados [1][2] -> corresponde ao conteúdo de cVariavel2

// aDados [1][3] -> corresponde ao conteúdo de cVariavel3

```

// aDados [2][1] -> corresponde ao conteúdo de cVariavel1
// aDados [2][2] -> corresponde ao conteúdo de cVariavel2
// aDados [2][3] -> corresponde ao conteúdo de cVariavel3

// aDados [3][1] -> corresponde ao conteúdo de cVariavel1
// aDados [3][2] -> corresponde ao conteúdo de cVariavel2
// aDados [3][3] -> corresponde ao conteúdo de cVariavel3

// Desta forma, o array aDados montando com uma estrutura de 03 linhas e 03 colunas, com
// o conteúdo definido por variáveis externas, mas com a mesma forma obtida com o uso do
// comando: aDados := ARRAY(3,3).

```

## ACLONE()

A função ACLONE() realiza a cópia dos elementos de um array para outro array integralmente.

Sintaxe: AADD(aArray)

Parâmetros

aArray	Array pré-existente que terá seu conteúdo copiado para o array especificado.
--------	--

Exemplo:

```

// Utilizando o array aDados, utilizado no exemplo da função AADD()
aItens := ACLONE(aDados).

// Neste ponto, o array aItens possui exatamente a mesma estrutura e informações do array
// aDados.

```



*Fique  
atento*

Por ser uma estrutura de memória, um array não pode ser simplesmente copiado para outro array, com a utilização de uma atribuição simples (“:=”).

Para mais informações sobre a necessidade de utilizar o comando **ACLONE()**, verifique o tópico 6.1.3 – Cópia de Arrays.

## ADEL()

A função ADEL() permite a exclusão de um elemento do array. Ao efetuar a exclusão de um elemento, todos os demais são reorganizados de forma que a última posição do array passará a ser nula.

Sintaxe: ADEL(aArray, nPosição)

Parâmetros

aArray	Array do qual deseja-se remover uma determinada posição.
nPosição	Posição do array que será removida.

Exemplo:

```
// Utilizando o array aItens do exemplo da função ACLONE() temos:
```

```
ADEL(aItens,1) // Será removido o primeiro elemento do array aItens.
```

```
// Neste ponto, o array aItens continua com 03 elementos, onde:
```

```
// aItens[1] -> antigo aItens[2], o qual foi reordenado como efeito da exclusão do item 1.
```

```
// aItens[2] -> antigo aItens[3], o qual foi reordenado como efeito da exclusão do item 1.
```

```
// aItens[3] -> conteúdo nulo, por se tratar do item excluído.
```

## ASIZE()

A função ASIZE permite a redefinição da estrutura de um array pré-existente, adicionando ou removendo itens do mesmo.

Sintaxe: ASIZE(aArray, nTamanho)

Parâmetros

aArray	Array pré-existente que terá sua estrutura redimensionada.
nTamanho	Tamanho com o qual se deseja redefinir o array. Se o tamanho for menor do que o atual, serão removidos os elementos do final do array, já se o tamanho for maior do que o atual serão inseridos itens nulos ao final do array.

Exemplo:

```
// Utilizando o array aItens, o qual teve um elemento excluído pelo uso da função ADEL()
```

```
ASIZE(aItens,Len(aItens-1)).
```

```
// Nesse ponto o array aItens possui 02 elementos, ambos com conteúdos válidos.
```



Dica

Utilizar a função ASIZE(), após o uso da função ADEL(), é uma prática recomendada e evita que seja acessada uma posição do array com um conteúdo inválido para a aplicação em uso.

## ASORT()

A função ASORT() permite que os itens de um array sejam ordenados a partir de um critério pré-estabelecido.

Sintaxe: ASORT(aArray, nInicio, nItens, bOrdem)

Parâmetros

aArray	Array pré-existente que terá seu conteúdo ordenado utilizando um critério estabelecido.
nInicio	Posição inicial do array para início da ordenação. Se não for informado, o array será ordenado a partir de seu primeiro elemento.
nItens	Quantos itens, a partir da posição inicial deverão ser ordenados. Se não informado, serão ordenados todos os elementos do array.
bOrdem	O bloco de código que permite a definição do critério de ordenação do array. Se o bOrdem não for informado, será utilizado o critério ascendente.



Dica

Um bloco de código é basicamente uma função escrita em linha. Dessa forma, sua estrutura deve “suportar” todos os requisitos de uma função, os quais são por meio da análise e interpretação de parâmetros recebidos, executar um processamento e fornecer um retorno.

Com base nesse requisito, é possível definir um bloco de código com a estrutura abaixo:

bBloco := { |xPar1, xPar2, ... xParZ| Ação1, Ação2, AçãoZ } , aonde:

|| -> define o intervalo onde estão compreendidos os parâmetros

Ação Z-> expressão que será executadas pelo bloco de código

Ação1... AçãoZ -> intervalo de expressões que serão executadas pelo bloco de código, no formato de lista de expressões.

Retorno -> resultado da última ação executada pelo bloco de código, no caso

AçãoZ.

Para maiores detalhes sobre a estrutura e utilização de blocos de código, consulte o tópico 6.2 – Listas de Expressões e Blocos de código.

#### Exemplo 01 – Ordenação ascendente

```
aAlunos := { “Mauren”, “Soraia”, “Andréia” }
```

```
aSort(aAlunos)
```

```
// Nesse ponto, os elementos do array aAlunos serão {“Andréia”, “Mauren”, “Soraia”}
```

## Exemplo 02 – Ordenação descendente

```
aAlunos := { "Mauren", "Soraia", "Andréia"}
```

```
bOrdem := { |x,y| x > y }
```

```
// Durante a execução da função aSort(), a variável "x" receberá o conteúdo do item que está  
// posicionado. Como o item que está posicionado é a posição aAlunos[x] e aAlunos[x] ->  
// string contendo o nome de um aluno, é possível substituir "x" por cNomeAtu.  
// A variável "y" receberá o conteúdo do próximo item a ser avaliado, e usando a mesma  
// analogia de "x", é possível substituir "y" por cNomeProx. Dessa forma o bloco de código  
// bOrdem pode ser re-escrito como:
```

```
bOrdem := { |cNomeAtu, cNomeProx| cNomeAtu > cNomeProx }
```

```
aSort(aAlunos,,bOrdem)
```

```
// Nesse ponto, os elementos do array aAlunos serão {"Soraia", "Mauren", "Andréia"}
```

## ASCAN()

A função ASCAN() permite que seja identificada a posição do array que contém uma determinada informação, através da análise de uma expressão descrita em um bloco de código.

Sintaxe: ASCAN(aArray, bSeek)

Parâmetros

aArray	Array pré-existente no qual deseja identificar a posição que contém a informação pesquisada.
bSeek	Bloco de código que configura os parâmetros da busca ser realizada.

Exemplo:

```
aAlunos := {"Márcio", "Denis", "Arnaldo", "Patrícia"}
```

```
bSeek := {|x| x == "Denis"}
```

```
nPosAluno := aScan(aAlunos,bSeek) // retorno esperado      2
```



*Dica*

Durante a execução da função aScan, a variável “x” receberá o conteúdo do item que está posicionado no momento, no caso aAlunos[x]. Como aAlunos[x] é uma posição do array que contém o nome do aluno, “x” poderia ser renomeada para cNome, e a definição do bloco bSeek poderia ser re-escrita como:

```
bSeek := {|cNome| cNome == "Denis"}
```



Dica

Na definição dos programas é sempre recomendável utilizar variáveis com nomes significativos, desta forma os blocos de código não são exceção.

Sempre opte por analisar como o bloco de código será utilizado e ao invés de “x”, “y” e similares, defina os parâmetros com nomes que representem seu conteúdo. Será mais simples o seu entendimento e o entendimento de outros que forem analisar o código escrito.

## AINS()

---

A função AINS() permite a inserção de um elemento no array especificado, em qualquer ponto da estrutura do mesmo, diferindo dessa forma da função AADD(), a qual sempre insere um novo elemento ao final da estrutura já existente.

Sintaxe: AINS(aArray, nPosicao)

Parâmetros

aArray	Array pré-existente no qual desejasse inserir um novo elemento.
nPosicao	Posição na qual o novo elemento será inserido.

Exemplo:

```
aAlunos := {"Edson", "Robson", "Renato", "Tatiana"}
```

```
AINS(aAlunos,3)
```

```
// Nesse ponto o array aAlunos terá o seguinte conteúdo:
```

```
// {"Edson", "Robson", nulo, "Renato", "Tatiana"}
```



*Fique  
atento*

Similar ao efeito da função ADEL(), o elemento inserido no array pela função AINS() terá um conteúdo nulo, sendo necessário tratá-lo após a realização deste comando.

## Manipulação de blocos de código

### EVAL()

---

A função EVAL() é utilizada para avaliação direta de um bloco de código, utilizando as informações disponíveis no mesmo de sua execução. Esta função permite a definição e passagem de diversos parâmetros que serão considerados na interpretação do bloco de código.

Sintaxe: EVAL(bBloco, xParam1, xParam2, xParamZ)

Parâmetros

bBloco	Bloco de código que será interpretado.
xParamZ	Parâmetros que serão passados ao bloco de código. A partir da passagem do bloco, todos os demais parâmetros da função serão convertidos em parâmetros para a interpretação do código.

Exemplo:

```
nInt := 10  
bBloco := { |N| x:= 10, y:= x*N, z:= y/(x*N) }  
  
nValor := EVAL(bBloco, nInt)
```

// O retorno será dado pela avaliação da ultima ação da lista de expressões, no caso “z”.

// Cada uma das variáveis definidas, em uma das ações da lista de expressões, fica disponível  
// para a próxima ação.

// Desta forma temos:

```
// N    recebe nInt como parâmetro (10)  
// X    tem atribuído o valor 10 (10)  
// Y    resultado da multiplicação de X por N (100)  
// Z    resultado da divisão de Y pela multiplicação de X por N ( 100 / 100)      1
```

## DBEVAL()

A função DBEval() permite que todos os registros de uma determinada tabela sejam analisados, e para cada registro será executado o bloco de código definido.

Sintaxe: Array(bBloco, bFor, bWhile)

Parâmetros

bBloco	Bloco de código principal, contendo as expressões que serão avaliadas para cada registro do alias ativo.
bFor	Condição para continuação da análise dos registros, com o efeito de uma estrutura For ... Next.
bWhile	Condição para continuação da análise dos registros, com o efeito de uma estrutura While ... End.

### Exemplo 01

Considerando o trecho de código abaixo:

```
dbSelectArea("SX5")
dbSetOrder(1)
dbGotop()
```

```
While !Eof() .And. X5_FILIAL == xFilial("SX5") .And.; X5_TABELA <= mv_par02
```

```
nCnt++
dbSkip()
```

```
End
```

O mesmo pode ser re-escrito com o uso da função DBEVAL():

```
dbEval( {|x| nCnt++ },,{||X5_FILIAL==xFilial("SX5") .And. X5_TABELA<=mv_par02})
```

### Exemplo 02

Considerando o trecho de código abaixo:

```
dbSelectArea("SX5")
dbSetOrder(1)
dbGotop()
```

```
While !Eof() .And. X5_TABELA == cTabela
```

```
    AADD(aTabela,{X5_CHAVE, Capital(X5_DESCRI)})
```

```
    dbSkip()
```

```
End
```

Exemplo 02 (continuação):

O mesmo pode ser re-escrito com o uso da função DBEVAL():

```
dbEval({|| aAdd(aTabela,{X5_CHAVE,Capital(X5_DESCRI)})},,{|| X5_TABELA==cTabela})
```



*Fique  
atento*

Na utilização da função DBEVAL(), deve ser informado apenas um dos dois parâmetros: bFor ou bWhile.

## AEVAL()

A função AEVAL() permite que todos os elementos de um determinada array sejam analisados e para cada elemento será executado o bloco de código definido.

Sintaxe: AEVAL(aArray, bBloco, nInicio, nFim)

Parâmetros

aArray	Array que será avaliado na execução da função
bBloco	Bloco de código principal, contendo as expressões que serão avaliadas para cada elemento do array informado
nInicio	Elemento inicial do array, a partir do qual serão avaliados os blocos de código
nFim	Elemento final do array, até o qual serão avaliados os blocos de código

Exemplo 01:

Considerando o trecho de código abaixo:

```
AADD(aCampos,"A1_FILIAL")
AADD(aCampos,"A1_COD")
SX3->(dbSetOrder(2))
For nX:=1 To Len(aCampos)
    SX3->(dbSeek(aCampos[nX]))
    aAdd(aTitulos,AllTrim(SX3->X3_TITULO))
Next nX
```

O mesmo pode ser re-escrito com o uso da função AEVAL():

```
aEval(aCampos,{|x| SX3->(dbSeek(x)),IIF(Found(), AAdd(aTitulos,;
    AllTrim(SX3->X3_TITULO))))})
```



## Manipulação de strings

---

### ALLTRIM()

Retorna uma string sem os espaços à direita e à esquerda, referente ao conteúdo informado como parâmetro.

A função ALLTRIM() implementa as ações das funções RTRIM (“right trim”) e LTRIM (“left trim”).

Sintaxe: ALLTRIM(cString)

Parâmetros

cString	String que será avaliada para remoção dos espaços à direita e à esquerda.
---------	---

Exemplo:

```
cNome := ALLTRIM(SA1->A1_NOME)
```

```
MSGINFO("Dados do campo A1_NOME:" + CRLF
```

```
    "Tamanho:" + CVALTOCHAR(LEN(SA1->A1_NOME)) + CRLF
```

```
    "Texto:" + CVALTOCHAR(LEN(cNome)))
```

---

### ASC()

Converte uma informação caractere em seu valor, de acordo com a tabela ASCII.

Sintaxe: ASC(cCaractere)

Parâmetros

cCaractere	Caractere que será consultado na tabela ASCII.
------------	--

Exemplo:

```
USER FUNCTION NoAcento(Arg1)
Local nConta := 0
Local cLetra := ""
Local cRet    := ""
Arg1 := Upper(Arg1)
For nConta:= 1 To Len(Arg1)
    cLetra := SubStr(Arg1, nConta, 1)
    Do Case
        Case (Asc(cLetra) > 191 .and. Asc(cLetra) < 198) .or.;
            (Asc(cLetra) > 223 .and. Asc(cLetra) < 230)
                cLetra := "A"
        Case (Asc(cLetra) > 199 .and. Asc(cLetra) < 204) .or.;
            (Asc(cLetra) > 231 .and. Asc(cLetra) < 236)
                cLetra := "E"
        Case (Asc(cLetra) > 204 .and. Asc(cLetra) < 207) .or.;
            (Asc(cLetra) > 235 .and. Asc(cLetra) < 240)
                cLetra := "I"
        Case (Asc(cLetra) > 209 .and. Asc(cLetra) < 215) .or.;
            (Asc(cLetra) == 240) .or. (Asc(cLetra) > 241 .and. Asc(cLetra) < 247)
                cLetra := "O"
        Case (Asc(cLetra) > 216 .and. Asc(cLetra) < 221) .or.;
            (Asc(cLetra) > 248 .and. Asc(cLetra) < 253)
                cLetra := "U"
        Case Asc(cLetra) == 199 .or. Asc(cLetra) == 231
            cLetra := "C"
```

EndCase

cRet := cRet+cLetra

Next

Return UPPER(cRet)

## AT()

Retorna à primeira posição de um caractere ou string dentro de outra string especificada.

Sintaxe: AT(cCaractere, cString )

Parâmetros

cCaractere	Caractere ou string que se deseja verificar.
cString	String na qual será verificada a existência do conteúdo de cCaractere.

Exemplo:

```
STATIC FUNCTION NOMASCARA(cString,cMascara,nTamanho)
```

```
LOCAL cNoMascara    := ""
```

```
LOCAL nX           := 0
```

```
IF !Empty(cMascara) .AND. AT(cMascara,cString) > 0
```

```
    FOR nX := 1 TO Len(cString)
```

```
        IF !(SUBSTR(cString,nX,1) $ cMascara)
```

```
            cNoMascara += SUBSTR(cString,nX,1)
```

```
        ENDIF
```

```
    NEXT nX
```

```
    cNoMascara := PADR(ALLTRIM(cNoMascara),nTamanho)
```

```
ELSE
```

```
    cNoMascara := PADR(ALLTRIM(cString),nTamanho)
```

```
ENDIF
```

```
RETURN cNoMascara
```

## CHR()

Converte um valor número, referente a uma informação da tabela ASCII, no caractere que esta informação representa.

Sintaxe: CHR(nASCII)

Parâmetros

nASCII	Código ASCII do caractere.
--------	----------------------------

Exemplo:

```
#DEFINE CRLF CHR(13)+CHR(10) // FINAL DE LINHA
```

## LEN()

Retorna o tamanho da string especificada no parâmetro.

Sintaxe: LEN(cString)

Parâmetros

cString	String que será avaliada.
---------	---------------------------

Exemplo:

```
cNome := ALLTRIM(SA1->A1_NOME)
```

```
MSGINFO("Dados do campo A1_NOME:" + CRLF
```

```
    "Tamanho:" + CVALTOCHAR(LEN(SA1->A1_NOME)) + CRLF
```

```
    "Texto:" + CVALTOCHAR(LEN(cNome)))
```

## LOWER()

Retorna uma string com todos os caracteres minúsculos, tendo como base a string passada como parâmetro.

Sintaxe: LOWER(cString)

Parâmetros

cString	String que será convertida para caracteres minúsculos.
---------	--

Exemplo:

```
cTexto := "ADVPL"
```

```
MSGINFO("Texto:"+LOWER(cTexto))
```

## RAT()

Retorna a última posição de um caracter ou string dentro de outra string especificada.

Sintaxe: RAT(cCaractere, cString)

Parâmetros

cCaractere	Caractere ou string que se deseja verificar
cString	String na qual será verificada a existência do conteúdo de cCaractere

## STUFF()

Permite substituir um conteúdo caractere em uma string já existente, especificando a posição inicial para esta adição e o número de caracteres que serão substituídos.

Sintaxe: STUFF(cString, nPosInicial, nExcluir, cAdicao)

Parâmetros

--	--

Exemplo:

```
cLin := Space(100)+cEOL // Cria a string base
```

```
cCpo := PADR(SA1->A1_FILIAL,02) // Informação que será armazenada na string
```

```
cLin := Stuff(cLin,01,02,cCpo) // Substitui o conteúdo de cCpo na string base
```

## SUBSTR()

Retorna parte do conteúdo de uma string especificada, de acordo com a posição inicial deste conteúdo na string e a quantidade de caracteres que deverá ser retornada a partir daquele ponto (inclusive).

Sintaxe: SUBSTR(cString, nPosInicial, nCaracteres)

Parâmetros

cString	String que se deseja verificar
nPosInicial	Posição inicial da informação que será extraída da string
nCaracteres	Quantidade de caracteres que deverão ser retornados a partir daquele ponto (inclusive)

Exemplo:

cCampo := "A1\_NOME"

nPosUnder := AT(cCampo)

cPrefixo := SUBSTR(cCampo,1, nPosUnder) // "A1\_"

## UPPER()

Retorna uma string com todos os caracteres maiúsculos, tendo como base a string passada como parâmetro.

Sintaxe: UPPER(cString)

Parâmetros

cString	String que será convertida para caracteres maiúsculos.
---------	--

Exemplo:

```
cTexto := "advpl"
```

```
MSGINFO("Texto:"+LOWER(cTexto))
```

## Manipulação de variáveis numéricas

### ABS()

Retorna um valor absoluto (independente do sinal), com base no valor especificado no parâmetro.

Sintaxe: ABS(nValor)

Parâmetros

nValor	Valor que será avaliado
--------	-------------------------

Exemplo:

```
nPessoas := 20
```

```
nLugares := 18
```

```
IF nPessoas < nLugares
```

```
    MSGINFO("Existem "+CVALTOCHAR(nLugares- nPessoas)+" disponíveis")
```

```
ELSE
```

```
    MSGSTOP("Existem "+CVALTOCHAR(ABS(nLugares- nPessoas))+ "faltando")
```

```
ENDIF
```

## INT()

Retorna a parte inteira de um valor especificado no parâmetro.

Sintaxe: INT(nValor)

Parâmetros

nValor	Valor que será avaliado
--------	-------------------------

Exemplo:

```
STATIC FUNCTION COMPRAR(nQuantidade)
```

```
LOCAL nDinheiro := 0.30
```

```
LOCAL nPrcUnit := 0.25
```

```
IF nDinheiro >= (nQuantidade*nPrcUnit)
```

```
    RETURN nQuantidade
```

```
ELSEIF nDinheiro > nPrcUnit
```

```
    nQuantidade := INT(nDinheiro / nPrcUnit)
```

```
ELSE
```

```
    nQuantidade := 0
```

```
ENDIF
```

```
RETURN nQuantidade
```

## NOROUND()

Retorna um valor, truncando a parte decimal do valor especificado no parâmetro, de acordo com a quantidade de casas decimais solicitadas.

Sintaxe: NOROUND(nValor, nCasas)

Parâmetros

nValor	Valor que será avaliado
nCasas	Número de casas decimais válidas. A partir da casa decimal especificada os valores serão desconsiderados

Exemplo:

nBase := 2.985

nValor := NOROUND(nBase,2) 2.98

## ROUND()

Retorna um valor, arredondando a parte decimal do valor especificado no parâmetro, de acordo com a quantidades de casas decimais solicitadas, utilizando o critério matemático.

Sintaxe: ROUND(nValor, nCasas)

Parâmetros

nValor	Valor que será avaliado
nCasas	Número de casas decimais válidas. As demais casas decimais sofrerão o arredondamento matemático, aonde:  Se $nX \leq 4$ = 0, senão +1 para a casa decimal superior

Exemplo:

nBase := 2.985

nValor := ROUND(nBase,2) = 2.99



Anotações

---

---

---

---

## Manipulação de arquivos

### SELECT()

Determina o número de referência de um determinado alias em um ambiente de trabalho. Caso o alias especificado não esteja em uso no ambiente, será retornado o valor 0 (zero).

Sintaxe: Select(cArea)

Parâmetros

cArea	Nome de referência da área de trabalho a ser verificada
-------	---

Exemplo:

```
nArea := Select("SA1")
```

```
ALERT("Referência do alias SA1: "+STRZERO(nArea,3)) // 10 (proposto)
```

### DBGOTO()

Move o cursor da área de trabalho ativa para o record number (recno) especificado, realizando um posicionamento direto, sem a necessidade uma busca (seek) prévio.

Sintaxe: DbGoto(nRecno)

Parâmetros

nRecno	Record number do registro a ser posicionado
--------	---

Exemplo:

DbSelectArea("SA1")

DbGoto(100) // Posiciona no registro 100

IF !EOF() // Se a área de trabalho não estiver em final de arquivo

MsgInfo("Você está no cliente:" + A1\_NOME)

ENDIF



Anotações

---

---

---

---

## DBGOTOP()

Move o cursor da área de trabalho ativa para o primeiro registro lógico.

Sintaxe: DbGoTop()

Parâmetros

Nenhum	.
--------	---

Exemplo:

```
nCount := 0 // Variável para verificar quantos registros há no intervalo
DbSelectArea("SA1")
DbSetOrder(1) // A1_FILIAL + A1_COD + A1_LOJA
DbGoTop()

While !BOF() // Enquanto não for o início do arquivo
    nCount++ // Incrementa a variável de controle de registros no intervalo
    DbSkip(-1)
End
```

```
MsgInfo("Existem :" + STRZERO(nCount,6) + " registros no intervalo").
```

```
// Retorno esperado :000001, pois o DbGoTop posiciona no primeiro registro.
```

## DBGOBUTTON()

Move o cursor da área de trabalho ativa para o último registro lógico.

Sintaxe: DbGoBotton()

Parâmetros

Nenhum	.
--------	---

Exemplo:

```
nCount := 0 // Variável para verificar quantos registros há no intervalo
DbSelectArea("SA1")
DbSetOrder(1) // A1_FILIAL + A1_COD + A1_LOJA
DbGoBotton()

While !EOF() // Enquanto não for o início do arquivo
    nCount++ // Incrementa a variável de controle de registros no intervalo
    DbSkip(1)
End

MsgInfo("Existem :" + STRZERO(nCount,6) + " registros no intervalo").

// Retorno esperado :000001, pois o DbGoBotton posiciona no último registro.
```

## DBSELECTAREA()

Define a área de trabalho especificada como sendo a área ativa. Todas as operações subsequentes que fizerem referência a uma área de trabalho para utilização, a menos que a área desejada seja informada explicitamente.

Sintaxe: DbSelectArea(nArea | cArea)

Parâmetros

nArea	Valor numérico que representa a área desejada, em função de todas as áreas já abertas pela aplicação, que pode ser utilizado ao invés do nome da área
cArea	Nome de referência da área de trabalho a ser selecionada

Exemplo 01: DbselectArea(nArea)

```
nArea := Select("SA1") // 10 (proposto)
```

DbSelectArea(nArea) // De acordo com o retorno do comando Select().

```
ALERT("Nome do cliente: "+A1_NOME) // Como o SA1 é o alias selecionado, os comandos
// a partir da seleção do alias compreendem que ele
// está implícito na expressão, o que causa o mesmo
// efeito de SA1->A1_NOME.
```

Exemplo 01: DbselectArea(cArea)

```
DbSelectArea("SA1") // Especificação direta do alias que deseja-se selecionar.
```

```
ALERT("Nome do cliente: "+A1_NOME) // Como o SA1 é o alias selecionado, os comandos
// a partir da seleção do alias compreendem que ele
// está implícito na expressão, o que causa o mesmo
// efeito de SA1->A1_NOME.
```

## DBSETORDER()

Define qual índice será utilizado pela área de trabalho ativa, ou seja, pela área previamente selecionada através do comando DbSelectArea(). As ordens disponíveis no Ambiente Protheus são aquelas definidas no SINdex /SIX, ou as ordens disponibilizadas por meio de índices temporários.

Sintaxe: DbSetOrder(nOrdem)

Parâmetros

nOrdem	Número de referência da ordem que deseja ser definida como ordem ativa para a área de trabalho
--------	--

Exemplo:

```
DbSelectArea("SA1")
```

```
DbSetOrder(1) // De acordo com o arquivo SIX -> A1_FILIAL+A1_COD+A1_LOJA
```

## DBORDERNICKNAME()

Define qual índice criado pelo usuário será utilizado. O usuário pode incluir os seus próprios índices e no momento da inclusão deve criar o NICKNAME para o mesmo.

Sintaxe: DbOrderNickName(NickName)

Parâmetros

NickName	NickName atribuído ao índice criado pelo usuário.
----------	---

Exemplo:

```
DbSelectArea("SA1")
```

```
DbOrderNickName("Tipo") // De acordo com o arquivo SIX -> A1_FILIAL+A1_TIPO
```

```
NickName: Tipo
```

## DBSEEK() E MSSEEK()

DbSeek(): Permite posicionar o cursor da área de trabalho ativo no registro com as informações especificadas na chave de busca, fornecendo um retorno lógico e indicando se o posicionamento foi efetuado com sucesso, ou seja, se a informação especificada, na chave de busca, foi localizada na área de trabalho.

Sintaxe: DbSeek(cChave, lSoftSeek, lLast)

Parâmetros

cChave	Dados do registro que se deseja localizar, de acordo com a ordem de busca previamente especificada pelo comando DbSetOrder(), ou seja, de acordo com o índice ativo no momento para a área de trabalho.
lSoftSeek	Define se o cursor ficará posicionado no próximo registro válido, em relação à chave de busca especificada, ou em final de arquivo, caso não seja encontrada exatamente a informação da chave. Padrão .F.
lLast	Define se o cursor será posicionado no primeiro ou no último registro de um intervalo, com as mesmas informações especificadas na chave. Padrão .F.

### Exemplo 01 – Busca exata

```
DbSelectArea("SA1")
```

```
DbSetOrder(1) // acordo com o arquivo SIX ->
```

```
A1_FILIAL+A1_COD+A1_LOJA
```

```
IF DbSeek("01" + "000001" + "02") // Filial: 01, Código: 000001, Loja: 02
```

```
    MsgInfo("Cliente localizado", "Consulta por cliente")
```

```
Else
```

```
    MsgAlert("Cliente não encontrado", "Consulta por cliente")
```

```
Endif
```

Exemplo 02 – Busca aproximada

```
DbSelectArea("SA1")
DbSetOrder(1) // acordo com o arquivo SIX -> A1_FILIAL+A1_COD+A1_LOJA

DbSeek("01" + "000001" + "02", .T.) // Filial: 01, Código: 000001, Loja: 02

// Exibe os dados do cliente localizado, o qual pode não ser o especificado na chave:

MsgInfo("Dados do cliente localizado: "+CRLF +;
        "Filial:"    + A1_FILIAL  + CRLF +;
        "Código:"   + A1_COD    + CRLF +;
        "Loja:"     + A1_LOJA   + CRLF +;
        "Nome:"     + A1_NOME   + CRLF, "Consulta por cliente")
```

**MsSeek():** Função desenvolvida pela área de Tecnologia da Microsiga, a qual possui as mesmas funcionalidades básicas da função DbSeek(), com a vantagem de não necessitar novamente do acesso da base de dados para localizar uma informação já utilizada pela thread (conexão) ativa.

Desta forma, a thread mantém em memória os dados necessários para reposicionar os registros já localizados através do comando DbSeek (no caso o Recno()), de forma que a aplicação pode simplesmente efetuar o posicionamento sem executar novamente a busca.

A diferença entre o DbSeek() e o MsSeek() é notada em aplicações com grande volume de posicionamentos, como relatórios, que necessitam referenciar diversas vezes o mesmo registro, durante uma execução.

### DBSKIP()

Move o cursor do registro posicionado para o próximo (ou anterior dependendo do parâmetro), em função da ordem ativa para a área de trabalho.

Sintaxe: DbSkip(nRegistros)

Parâmetros

nRegistros	Define em quantos registros o cursor será deslocado. Padrão	1
------------	---	---

#### Exemplo 01 – Avançando registros

```
DbSelectArea("SA1")
DbSetOrder(2) // A1_FILIAL + A1_NOME
DbGotop() // Posiciona o cursor no início da área de trabalho ativa.
```

```
While !EOF() // Enquanto o cursor da área de trabalho ativa não indicar fim de arquivo
    MsgInfo("Você está no cliente:" + A1_NOME)
    DbSkip()
End
```

Exemplo 02 – Retrocedendo registros

```
DbSelectArea("SA1")
```

```
DbSetOrder(2) // A1_FILIAL + A1_NOME
```

```
DbGoBottom() // Posiciona o cursor no final da área de trabalho ativa.
```

```
While !BOF() // Enquanto o cursor da área de trabalho ativa não indicar início de arquivo
```

```
    MsgInfo("Você está no cliente:" + A1_NOME)
```

```
    DbSkip(-1)
```

```
End
```

## DBSETFILTER()

Define um filtro para a área de trabalho ativa, o qual pode ser descrito na forma de um bloco de código ou através de uma expressão simples.

Sintaxe: DbSetFilter(bCondicao, cCondicao)

Parâmetros

bCondicao	Bloco que expressa a condição de filtro em forma executável
cCondicao	Expressão de filtro simples na forma de string

### Exemplo 01 – Filtro com bloco de código

```
bCondicao := {|| A1_COD >= “000001” .AND. A1_COD <= “001000”}
```

```
DbSelectArea(“SA1”)
```

```
DbSetOrder(1)
```

```
DbSetFilter(bCondicao)
```

```
DbGoBotton()
```

```
While !EOF()
```

```
    MsgInfo(“Você está no cliente:”+A1_COD)
```

```
    DbSkip()
```

```
End
```

```
// O último cliente visualizado deve ter o código menor do que “001000”.
```

Exemplo 02 – Filtro com expressão simples

```
cCondicao := “A1_COD >= ‘000001’ .AND. A1_COD <= ‘001000’”
```

```
DbSelectArea(“SA1”)
```

```
DbSetOrder(1)
```

```
DbSetFilter(,cCondicao)
```

```
DbGoBotton()
```

```
While !EOF()
```

```
    MsgInfo(“Você está no cliente:”+A1_COD)
```

```
    DbSkip()
```

```
End
```

```
// O último cliente visualizado deve ter o código menor do que “001000”.
```

## DBSTRUCT()

Retorna um array contendo a estrutura da área de trabalho (alias) ativo. A estrutura será um array bidimensional conforme abaixo:

I	No	Tip	Tama	Deci
D	me	o	nho	mais
*	ca	ca		
	mp	mp		
	o	o		

\*Índice do array

Sintaxe: DbStruct()

Parâmetros

Nenhum	.
--------	---

Exemplo:

```
cCampos := ""
DbSelectArea("SA1")
aStructSA1 := DbStruct()
```

```
FOR nX := 1 to Len(aStructSA1)
```

```
    cCampos += aStructSA1[nX][1] + "/"
```

```
NEXT nX
```

```
ALERT(cCampos)
```

## RECLOCK()

Efetua o travamento do registro posicionado na área de trabalho ativa, permitindo a inclusão ou alteração das informações do mesmo.

Sintaxe: RecLock(cAlias,lInclui)

Parâmetros

cAlias	Alias que identifica a área de trabalho que será manipulada.
lInclui	Define se a operação será uma inclusão (.T.) ou uma alteração (.F.).

### Exemplo 01 - Inclusão

```
DbSelectArea("SA1")
RecLock("SA1",.T.)
SA1->A1_FILIAL := xFilial("SA1") // Retorna a filial de acordo com as configurações do ERP.
SA1->A1_COD := "900001"
SA1->A1_LOJA := "01"
MsUnLock() // Confirma e finaliza a operação.
```

### Exemplo 02 - Alteração

```
DbSelectArea("SA1")
DbSetOrder(1) // A1_FILIAL + A1_COD + A1_LOJA
DbSeek("01" + "900001" + "01") // Busca exata

IF Found() // Avalia o retorno do último DbSeek realizado
  RecLock("SA1",.F.)
  SA1->A1_NOME := "CLIENTE CURSO ADVPL BÁSICO"
  SA1->A1_NREDUZ := "ADVPL BÁSICO"
  MsUnLock() // Confirma e finaliza a operação
```

ENDIF



Dica

A linguagem ADVPL possui variações da função RecLock(), as quais são:

RLOCK()  
DBRLOCK()

A sintaxe e a descrição destas funções estão disponíveis no Guia de Referência Rápido ao final deste material.

## MSUNLOCK()

Libera o travamento (lock) do registro posicionado, confirmando as atualizações efetuadas neste registro.

Sintaxe: MsUnLock()

Parâmetros

Nenhum	.
--------	---

Exemplo:

```
DbSelectArea("SA1")
DbSetOrder(1) // A1_FILIAL + A1_COD + A1_LOJA
DbSeek("01" + "900001" + "01") // Busca exata
```

```
IF Found() // Avalia o retorno do último DbSeek realizado
RecLock("SA1",.F.)
```

```
SA1->A1_NOME := "CLIENTE CURSO ADVPL BÁSICO"
```

```
SA1->A1_NREDUZ := "ADVPL BÁSICO"  
MsUnLock() // Confirma e finaliza a operação  
ENDIF
```



*Dica*

A linguagem ADVPL possui variações da função MsUnlock(), as quais são:

UNLOCK()

DBUNLOCK()

DBUNLOCKALL()

A sintaxe e a descrição destas funções estão disponíveis no Guia de Referência Rápido ao final deste material.

## SOFTLOCK()

Permite a reserva do registro posicionado na área de trabalho ativa de forma que outras operações, com exceção da atual, não possam atualizar este registro. Difere da função RecLock(), pois não gera uma obrigação de atualização, e pode ser sucedido por ele.

Na aplicação ERP Protheus, o SoftLock() é utilizado nos browses, antes da confirmação da operação de alteração e exclusão, pois neste momento a mesma ainda não foi efetivada, mas outras conexões não podem acessar aquele registro, pois o mesmo está em manutenção, o que implementa a integridade da informação.

Sintaxe: SoftLock(cAlias)

Parâmetros

cAlias	Alias de referência da área de trabalho ativa, para o qual o registro posicionado será travado.
--------	---

Exemplo:

cChave := GetCliente() // Função ilustrativa que retorna os dados de busca de um cliente.

DbSelectArea("SA1")

DbSetOrder(1)

DbSeek(cChave)

IF Found()

    SoftLock() // Reserva o registro localizado

    lConfirma := AlteraSA1() // Função ilustrativa que exibe os dados do registro

                        // posicionado e permite a alteração dos mesmos.

    IF lConfirma

        RecLock("SA1",.F.)

        GravaSA1() // Função ilustrativa que altera os dados conforme a AlertaSA1().

        MsUnLock() // Liberado o RecLock() e o SoftLock() do registro.

    Endif

Endif

## DBDELETE()

Efetua a exclusão lógica do registro posicionado na área de trabalho ativa, sendo necessária sua utilização em conjunto com as funções RecLock() e MsUnLock().

Sintaxe: DbDelete()

Parâmetros

Nenhum	.
--------	---

Exemplo:

```
DbSelectArea("SA1")
DbSetOrder(1) // A1_FILIAL + A1_COD + A1_LOJA
DbSeek("01" + "900001" + "01") // Busca exata
```

IF Found()

```
RecLock("SA1",.F.) // Define que será realizada uma alteração no registro posicionado.
DbDelete() // Efetua a exclusão lógica do registro posicionado.
MsUnLock() // Confirma e finaliza a operação.
ENDIF
```

## DBUSEAREA()

Define um arquivo de base de dados como uma área de trabalho disponível na aplicação.

Sintaxe: DbUseArea(lNovo, cDriver, cArquivo, cAlias, lComparilhado,;
lSoLeitura)

## Parâmetros

lNovo	Parâmetro opcional que permite, caso o cAlias especificado já esteja em uso, ser fechado antes da abertura do arquivo da base de dados.
cDriver	Driver que permita a aplicação manipular o arquivo de base de dados especificado. A aplicação ERP possui a variável _LOCALDRIVER, definida a partir das configurações do .ini do server da aplicação. Algumas chaves válidas: “DBFCDX”, “CTREECDX”, “DBFCDXAX”, “TOPCONN”.
cArquivo	Nome do arquivo de base de dados que será aberto com o alias especificado.
cAlias	Alias para referência do arquivos de base de dados pela aplicação.
lComparilhado	Se o arquivo poderá ser utilizado por outras conexões.
lSoLeitura	Se o arquivo poderá ser alterado pela conexão ativa.

Exemplo:

```
DbUserArea(.T., “DBFCDX”, “\SA1010.DBF”, “SA1DBF”, .T., .F.)  
DbSelectArea(“SA1DBF”)  
MsgInfo(“A tabela SA1010.DBF possui:” + STRZERO(RecCount(),6) + “ registros.”)  
DbCloseArea()
```

## DBCLOSEAREA()

Permite que um alias presente na conexão seja fechado, o que viabiliza novamente seu uso em outro operação. Este comando tem efeito apenas no alias ativo na conexão, sendo necessária sua utilização em conjunto com o comando DbSelectArea().

Sintaxe: DbCloseArea()

Parâmetros

Nenhum	.
--------	---

Exemplo:

```

DbUserArea(.T., "DBFCDX", "\SA1010.DBF", "SA1DBF", .T., .F.)
DbSelectArea("SA1DBF")
MsgInfo("A tabela SA1010.DBF possui:" + STRZERO(RecCount(),6) + " registros.")
DbCloseArea()

```

## Controle de numeração sequencial

**GETSXENUM()**

Obtém o número sequência do alias especificado no parâmetro, através da referência aos arquivos de sistema SXE/SXF ou ao servidor de numeração, quando esta configuração está habilitada no ambiente Protheus.

Sintaxe: GETSXENUM(cAlias, cCampo, cAliasSXE, nOrdem)

Parâmetros

cAlias	Alias de referência da tabela para a qual será efetuado o controle da numeração sequencial.
cCampo	Nome do campo no qual está implementado o controle da numeração.
cAliasSXE	Parâmetro opcional, quando o nome do alias nos arquivos de controle de numeração não é o nome convencional do alias para o sistema ERP.
nOrdem	Número do índice para verificar qual a próxima ocorrência do número.

---

**CONFIRMSXE()**

Confirma o número alocado através do último comando GETSXENUM().

Sintaxe: CONFIRMSXE(lVerifica)

Parâmetros

IVerifica	Verifica se o número confirmado não foi alterado, e se por consequência já existe na base de dados.
-----------	---

## ROLLBACKSXE()

---

Descarta o número fornecido pelo último comando GETSXENUM(), retornando a numeração disponível para outras conexões.

Sintaxe: ROLLBACKSXE()

Parâmetros

Nenhum	.
--------	---

## Validação

---

### EXISTCHAV()

Retorna .T. ou .F. se o conteúdo especificado existe no alias especificado. Se existir será exibido um help de sistema com um aviso informando da ocorrência.

A Função utilizada normalmente para verificar se um determinado código de cadastro já existe na tabela na qual a informação será inserida, como por exemplo o CNPJ no cadastro de clientes ou, fornecedores.

Sintaxe: ExistChav(cAlias, cConteudo, nIndice)

Parâmetros

cAlias	Alias de referência para a validação da informação
cConteudo	Chave a ser pesquisada, sem a filial

nIndice	Índice de busca para consulta da chave
---------	--

### EXISTCPO()

Retorna .T. ou .F. se o conteúdo especificado não existe no alias especificado. Caso não exista será exibido um help de sistema com um aviso informando da ocorrência.

Função utilizada normalmente para verificar se a informação digitada em um campo, a qual depende de outra tabela, realmente existe nesta outra tabela, como por exemplo o código de um cliente em um pedido de venda.

Sintaxe: ExistCpo(cAlias, cConteudo, nIndice)

Parâmetros

cAlias	Alias de referência para a validação da informação
cConteudo	Chave a ser pesquisada, sem a filial
nIndice	Índice de busca para consulta da chave

## NAOVAZIO()

---

Retorna .T. ou .F. se o conteúdo do campo posicionado no momento não está vazio.

Sintaxe: NaoVazio()

Parâmetros

Nenhum	.
--------	---

## NEGATIVO()

---

Retorna .T. ou .F. se o conteúdo digitado para o campo é negativo.

Sintaxe: Negativo()

Parâmetros

Nenhum	.
--------	---

## PERTENCE()

---

Retorna .T. ou .F. se o conteúdo digitado para o campo está contido na string definida como parâmetro da função. Normalmente utilizada em campos com a opção de combo, pois caso contrário seria utilizada a função ExistCpo().

Sintaxe: Pertence(cString)

Parâmetros

cString	String contendo as informações válidas que podem ser digitadas para um campo.
---------	---

## POSITIVO()

---

Retorna .T. ou .F. se o conteúdo digitado para o campo é positivo.

Sintaxe: Positivo()

Parâmetros

Nenhum	.
--------	---

### TEXTO()

Retorna .T. ou .F. se o conteúdo digitado para o campo contém apenas números ou alfanuméricos.

Sintaxe: Texto()

Parâmetros

Nenhum	.
--------	---

### VAZIO()

Retorna .T. ou .F. se o conteúdo do campo posicionado no momento está vazio.

Sintaxe: Vazio()

Parâmetros

Nenhum	.
--------	---

### Parâmetros

## GETMV0

Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Se o parâmetro não existir, será exibido um help do sistema informando a ocorrência.

Sintaxe: GETMV(cParametro)

Parâmetros

cParametro	Nome do parâmetro do sistema no SX6, sem a especificação da filial de sistema.
------------	--

### GETNEWPAR()

Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Se o parâmetro não existir, será exibido um help do sistema informando a ocorrência.

Difere do SuperGetMV() pois considera que o parâmetro pode não existir na versão atual do sistema, e por consequência não será exibida a mensagem de help.

Sintaxe: GETNEWPAR(cParametro, cPadrao, cFilial)

Parâmetros

cParametro	Nome do parâmetro do sistema no SX6 sem a especificação da filial de sistema.
cPadrao	Conteúdo padrão que será utilizado, caso o parâmetro não exista no SX6.
cFilial	Define para qual filial será efetuada a consulta do parâmetro. Padrão filial corrente da conexão.

## PUTMV()

Atualiza o conteúdo do parâmetro especificado no arquivo SX6, de acordo com as parametrizações informadas.

Sintaxe: PUTMV(cParametro, cConteudo)

Parâmetros

cParametro	Nome do parâmetro do sistema no SX6, sem a especificação da filial de sistema
cConteudo	Conteúdo que será atribuído ao parâmetro no SX6

## SUPERGETMV()

Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Caso o parâmetro não exista, será exibido um help do sistema informando a ocorrência.

Difere do GetMv(), pois os parâmetros consultados são adicionados em uma área de memória, que permite que em uma nova consulta não seja necessário acessar e pesquisar o parâmetro na base de dados.

Sintaxe: SUPERGETMV(cParametro , lHelp , cPadrao , cFilial)

Parâmetros

cParametro	Nome do parâmetro do sistema no SX6, sem a especificação da filial de sistema.
lHelp	Se será exibida a mensagem de Help, caso o parâmetro não seja encontrado no SX6.
cPadrao	Conteúdo padrão que será utilizado caso o parâmetro não exista no SX6.

cFilial	Define para qual filial será efetuada a consulta do parâmetro.Padrão filial corrente da conexão.
---------	--

## Componentes da interface visual

### **MSDIALOG()**

Define o componente MSDIALOG() que é utilizado como base para os demais componentes da interface visual, pois um componente MSDIALOG() é uma janela da aplicação.

Sintaxe:

```
DEFINE MSDIALOG oObjetoDLG TITLE cTitulo FROM nLinIni,nColIni TO nLiFim,nColFim OF  
oObjetoRef UNIDADE
```

#### Parâmetros

<b>oObjetoDLG</b>	Posição do objeto Say em função da janela em que ele será definido
<b>cTitulo</b>	Título da janela de diálogo
<b>nLinIni, nColIni</b>	Posição inicial em linha / coluna da janela
<b>nLiFim, nColFim</b>	Posição final em linha / coluna da janela
<b>oObjetoRef</b>	Objeto dialog no qual a janela será definida
<b>UNIDADE</b>	Unidade de medida das dimensões: PIXEL

Exemplo:

```
DEFINE MSDIALOG oDlg TITLE cTitulo FROM 000,000 TO 080,300 PIXEL  
ACTIVATE MSDIALOG oDlg CENTERED
```



Anotações

---

---

---

---

## MSGT()

Define o componente visual MSGT, o qual é utilizado para captura de informações digitáveis na tela da interface.

Sintaxe:

```
@ nLinha, nColuna MSGT VARIABEL SIZE nLargura,nAltura UNIDADE OF oObjetoRef F3 cF3  
VALID VALID WHEN WHEN PICTURE cPicture
```

### Parâmetros

nLinha, nColuna	Posição do objeto MsGet em função da janela em que ele será definido
VARIABEL	Variável da aplicação que será inculada ao objeto MsGet, que definirá suas características e na qual será armazenado o que for informado no campo
nLargura,nAltura	Dimensões do objeto MsGet para exibição do texto
UNIDADE	Unidade de medida das dimensões: PIXEL
oObjetoRef	Objeto dialog no qual o componente será definido
cF3	String que define a consulta padrão, a qual será vinculada ao campo
VALID	Função de validação para o campo.
WHEN	Condição para manipulação do campo, a qual pode ser diretamente .T. ou .F., ou uma variável ou uma chamada de função.
cPicture	String contendo a definição da Picture de digitação do campo.

Exemplo:

```
@ 010,050 MSGET cCGC  SIZE 55,11 OF oDlg PIXEL PICTURE "@R 99.999.999/9999-99";  
VALID !Vazio()
```



*Anotações*

---

---

---

---

## SAY()

Define o componente visual SAY, o qual é utilizado para exibição de textos em uma tela de interface.

Sintaxe:

```
@ nLinha, nColuna SAY cTexto SIZE nLargura,nAltura UNIDADE OF oObjetoRef
```

### Parâmetros

nLinha, nColuna	Posição do objeto Say em função da janela em que ele será definido
cTexto	Texto que será exibido pelo objeto Say
nLargura,nAltura	Dimensões do objeto Say para exibição do texto
UNIDADE	Unidade de medida das dimensões: PIXEL
oObjetoRef	Objeto dialog no qual o componente será definido

Exemplo:

```
@ 010,010 SAY cTexto SIZE 55, 07 OF oDlg PIXEL
```

## BUTTON()

Define o componente visual Button, o qual permite a inclusão de botões de operação na tela da interface, os quais serão visualizados somente com um texto simples para sua identificação.

Sintaxe: BUTTON()

```
@ nLinha,nColuna BUTTON cTexto SIZE nLargura,nAltura UNIDADE OF oObjetoRef  
ACTION AÇÃO
```

## Parâmetros

nLinha,nColuna	Posição do objeto Button em função da janela em que ele será definido
cTexto	String contendo o texto que será exibido no botão
nLargura,nAltura	Dimensões do objeto Button para exibição do texto
UNIDADE	Unidade de medida das dimensões: PIXEL
oObjetoRef	Objeto dialog no qual o componente será definido
AÇÃO	Função ou lista de expressões que define o comportamento do botão quando ele for utilizado

Exemplo:

```
010, 120 BUTTON "Confirmar" SIZE 080, 047 PIXEL OF oDlg;  
ACTION (nOpc := 1,oDlg:End())
```

## SBUTTON()

Define o componente visual SButton, que permite a inclusão de botões de operação na tela da interface, os quais serão visualizados dependendo da interface do sistema ERP utilizada somente com um texto simples para sua identificação, ou com uma imagem (BitMap) pré-definida.

Sintaxe: SBUTTON()

```
DEFINE SBUTTON FROM nLinha, nColuna TYPE N ACTION AÇÃO STATUS OF oObjetoRet
```

## Parâmetros

nLinha, nColuna	Posição do objeto sButton em função da janela em que ele será definido
TYPE N	Número que indica o tipo do botão (imagem) pré-definida que será utilizada

AÇÃO	Função ou lista de expressões que define o comportamento do botão quando ele for utilizado
STATUS	Propriedade de uso do botão: ENABLE ou DISABLE
oObjetoRet	Objeto dialog no qual o componente será definido

Exemplo:

DEFINE

SBUTTON FROM AxCadastro(cAlias, cTitulo, cVldExc, cVldAlt)  
020, 120 TYPE 2

ACTION (nOpc :=

2,oDlg:End());

ENABLE OF

oDlg

Visual dos  
diferentes  
tipos de  
botões  
disponíveis

s:



## Interfaces de cadastro

---

AXCADASTRO()

---

Sintaxe	
Descrição	O AxCadastro() é uma funcionalidade de cadastro simples, com poucas opções de customização.

## MBROWSE()

Sintaxe	MBrowse(nLin1, nCol1, nLin2, nCol2, cAlias)
Descrição	A Mbrowse() é uma funcionalidade de cadastro que permite a utilização de recursos mais aprimorados na visualização e manipulação das informações do sistema.

## AXPESQUI()

Função de pesquisa padrão em registros exibidos pelos browses do sistema, a qual posiciona o browse no registro pesquisado. Exibe uma tela que permite a seleção do índice a ser utilizado na pesquisa e a digitação das informações que compõe a chave de busca.

Sintaxe: AXPESQUI()

Parâmetros

Nenhum	.
--------	---



*Anotações*

---

---

---

---

## AXVISUAL()

Função de visualização padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

Sintaxe: AXVISUAL(cAlias, nReg, nOpc, aAcho, nColMens, cMensagem, cFunc,; aButtons, lMaximized )

### Parâmetros

cAlias	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada
nReg	Record number (recno) do registro posicionado no alias ativo
nOpc	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização).
aAcho	Vetor com nome dos campos que serão exibidos. Os campos de usuário sempre serão exibidos se não existir no parâmetro um elemento com a expressão "NOUSER"
nColMens	Parâmetro não utilizado
cMensagem	Parâmetro não utilizado.
cFunc	Função que deverá ser utilizada para carregar as variáveis que serão utilizadas pela Enchoice. Neste caso o parâmetro IVirtual é definido internamente pela AxFunction() executada como .T.
aButtons	Botões adicionais para a EnchoiceBar, no formato: aArray[n][1] -> Imagem do botão; aArray[n][2] -> bloco de código contendo a ação do botão; aArray[n][3] -> título do botão.
lMaximized	Indica se a janela deverá ser ou não maximizada.

## AXINCLUI()

Função de inclusão padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

Sintaxe: AxInclui(cAlias, nReg, nOpc, aAcho, cFunc, aCpos, cTudoOk, lF3,; cTransact, aButtons, aParam, aAuto, IVirtual, lMaximized)

## Parâmetros

cAlias	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada
nReg	Record number (recno) do registro posicionado no alias ativo
nOpc	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização)
aAcho	Vetor com nome dos campos que serão exibidos. Os campos de usuário sempre serão exibidos se não existir no parâmetro um elemento com a expressão "NOUSER"
cFunc	Função que deverá ser utilizada para carregar as variáveis que serão utilizadas pela Enchoice. Neste caso, o parâmetro IVirtual é definido internamente pela AxFunction(), executada como .T.
aCpos	Vetor com nome dos campos que poderão ser editados
cTudoOk	Função de validação de confirmação da tela. Não deve ser passada como Bloco de Código, mas pode ser passada como uma lista de expressões, desde que a última ação efetue um retorno lógico:  “(Func1(), Func2(), ..., FuncX(), .T. )”
lF3	Indica se a enchoice está sendo criada em uma consulta F3 para utilizar variáveis de memória
cTransact	Função que será executada dentro da transação da AxFunction()
aButtons	Botões adicionais para a EnchoiceBar, no formato:  aArray[n][1] -> Imagem do botão aArray[n][2] -> bloco de código contendo a ação do botão aArray[n][3] -> título do botão
aParam	Funções para execução em pontos pré-definidos da AxFunction(), conforme abaixo:  aParam[1] := Bloco de código que será processado antes da exibição da interface. aParam[2] := Bloco de código para processamento na validação da confirmação. aParam[3] := Bloco de código que será executado dentro da transação da

	AxFunction(). aParam[4] := Bloco de código que será executado fora da transação da AxFunction().
aAuto	Array no formato utilizado pela funcionalidade MsExecAuto(). Caso seja informado este array, não será exibida a tela de interface, e será executada a função EnchAuto().  aAuto[n][1] := Nome do campo aAuto[n][2] := Conteúdo do campo aAuto[n][3] := Validação que será utilizada em substituição as validações do SX3.
IVirtual	Indica se a Enchoice() chamada pela AxFunction() utilizará variáveis de memória ou os campos da tabela na edição
IMaximized	Indica se a janela deverá ser ou não maximizada

## AXALTERA()

Função de alteração padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

Sintaxe: AXALTERA(cAlias, nReg, nOpc, aAcho, aCpos, nColMens, cMensagem,; cTudoOk, cTransact, cFunc, aButtons, aParam, aAuto, IVirtual, IMaximized)

### Parâmetros

Vide documentação de parâmetros da função AxInclui().

## **AXDELETA()**

---

Função de exclusão padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

Sintaxe: AXDELETA(cAlias, nReg, nOpc, cTransact, aCpos, aButtons, aParam,; aAuto, lMaximized)

### Parâmetros

cAlias	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada
nReg	Record number (recno) do registro posicionado no alias ativo
nOpc	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização)
cTransact	Função que será executada dentro da transação da AxFunction()
aCpos	Vetor com nome dos campos que poderão ser editados
aButtons	Botões adicionais para a EnchoiceBar, no formato: aArray[n][1] -> Imagem do botão aArray[n][2] -> bloco de código contendo a ação do botão aArray[n][3] -> título do botão
aParam	Funções para execução em pontos pré-definidos da AxFunction(), conforme abaixo: aParam[1] := Bloco de código que será processado antes da exibição da interface. aParam[2] := Bloco de código para processamento na validação da confirmação. aParam[3] := Bloco de código que será executado dentro da transação da AxFunction(). aParam[4] := Bloco de código que será executado fora da transação da AxFunction().
aAuto	Array no formato utilizado pela funcionalidade MsExecAuto(). Caso seja informado este array, não será exibida a tela de interface, e será executada a função EnchAuto(). aAuto[n][1] := Nome do campo aAuto[n][2] := Conteúdo do campo

	aAuto[n][3] := Validação que será utilizada em substituição as validações do SX3.
lMaximized	Indica se a janela deverá ser ou não maximizada

## Funções visuais para aplicações

---

**ALERT()**

Sintaxe: AVISO(cTexto)

Parâmetros

cTexto	Texto a ser exibido



---

**AVISO()**

Sintaxe: AVISO(cTitulo, cTexto, aBotoes, nTamanho)

Retorno: numérico indicando o botão selecionado.

Parâmetros

cTitulo	Título da janela
cTexto	Texto do aviso
aBotoes	Array simples (vetor) com os botões de opção
nTamanho	Tamanho (1,2 ou 3)



## FORMBACTH()

Sintaxe: FORMBATCH(cTitulo, aTexto, aBotoes, bValid, nAltura, nLargura )

Parâmetros

cTitulo	Título da janela
aTexto	Array simples (vetor) contendo cada uma das linhas de texto que serão exibidas no corpo da tela
aBotoes	Array com os botões do tipo SBUTTON(), com a seguinte estrutura:  {nTipo,lEnable,{   Ação() }}  
bValid	(opcional) Bloco de validação da janela
nAltura	(opcional) Altura em pixels da janela
nLargura	(opcional) Largura em pixels da janela
	

## MSGFUNCTIONS()

Sintaxe: MSGALERT(cTexto, cTitulo)

Sintaxe: MSGINFO(cTexto, cTitulo)

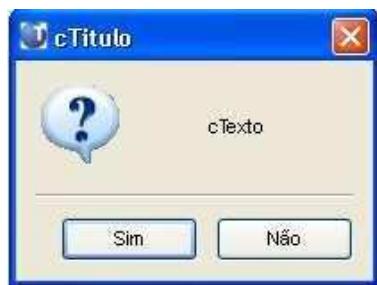
Sintaxe: MSGSTOP(cTexto, cTitulo)

Sintaxe: MSGYESNO(cTexto, cTitulo)

### Parâmetros

cTexto	Texto a ser exibido como mensagem
cTitulo	Título da janela de mensagem
MSGALERT	
MSGINFO	
MSGSTOP	

MSGYESNO



## Funções ADVPL para aplicações

---

### GETAREA()

Função utilizada para proteger o ambiente ativo no momento de algum processamento específico. Para salvar uma outra área de trabalho (alias) que não o ativo, a função GetArea() deve ser executada dentro do alias: ALIAS->(GetArea()).

Sintaxe: GETAREA()

Retorno: Array contendo {Alias(),IndexOrd(),Recno()}

Parâmetros

Nenhum	
--------	--

### RESTAREA()

---

Função utilizada para devolver a situação do ambiente salvo, através do comando GETAREA(). Deve-se observar que a última área restaurada é a área que ficará ativa para a aplicação.

Sintaxe: RESTAREA(aArea)

Parâmetros

aArea	Array contendo: {cAlias, nOrdem, nRecno}, normalmente gerado pelo uso da função GetArea().
-------	--

Exemplo:

```
// ALIAS ATIVO ANTES DA EXECUÇÃO DA ROTINA      SN3
```

```
User Function XATF001()
```

```
LOCAL cVar
```

```
LOCAL aArea := GetArea()
```

```
LOCAL lRet := .T.
```

```
cVar := &(ReadVar())

dbSelectArea("SX5")
IF !dbSeek(xFilial()+"Z1"+cVar)

    cSTR0001 := "REAV - Tipo de Reavaliacao"
    cSTR0002 := "Informe um tipo de reavalicao valido"
    cSTR0003 := "Continuar"
    Aviso(cSTR0001,cSTR0002,{cSTR0003},2)
    lRet := .F.

ENDIF

RestArea(aArea)
Return( lRet )
```

## **REFERÊNCIAS BIBLIOGRÁFICAS**

---

Referências bibliográficas

Gestão empresarial com ERP

Ernesto Haberkorn, 2006

Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados

Forbellone, André Luiz Villar - MAKRON, 1993

Introdução à Programação - 500 Algoritmos Resolvidos

Anita Lopes, Guto Garcia – CAMPUS / ELSEVIER, 2002

Apostila de Treinamento - ADVPL

Educação corporativa

Apostila de Treinamento – Introdução à programação

Educação corporativa

Apostila de Treinamento – Boas Práticas de Programação

Inteligência Protheus e Fábrica de Software

Curso Básico de Lógica de Programação

Paulo Sérgio de Moraes – PUC Campinas

TDN – TOTVS Developer Network

Microsiga Software S.A.

Materiais diversos de colaboradores Totvs

Colaboradores Totvs

