Protheus 10

Educação Corporativa

Programação em ADVPL Completo

microsiga

Matriz – Av. Braz Leme, 1.717 – 02511-000 – São Paulo – SP – Brasil.
Tel.: 55 (11) 3981-7001 www.microsiga.com.br

ESTRUTURA DO TREINAMENTO

MÓDULO 01: Introdução à programação *11*

1. Lógica de Programação e Algoritmos *11*

1.1. Lógica de Programação **11**

1.2. Desenvolvendo algoritmos **12**

1.2.1. Estudando algoritmos 13

1.2.2. Teste de mesa 15

2. Estruturas de programação *16*

2.1. Diagrama de bloco **16**

2.2. Estruturas de decisão e repetição **19**

2.2.1. Estruturas de decisão 19

2.2.2. Estruturas de repetição 22

MÓDULO 02: A linguagem ADVPL *24*

3. Estrutura de um Programa ADVPL *26*

3.1. Áreas de um Programa ADVPL **28**

4. Declaração e Atribuição de Variáveis *31*

4.1. Tipo de Dados **31**

4.2. Declaração de variáveis **32**

4.3. Escopo de variáveis **33**

4.4. Entendendo a influência do escopo das variáveis **37**

4.5. Operações com Variáveis **38**

4.5.1. Atribuição de variáveis 38

4.5.2. Operadores da linguagem ADVPL 39

4.5.3. Operação de Macro Substituição 44

4.5.4. Funções de manipulação de variáveis 45

5. Estruturas básicas de programação *51*

5.1. Estruturas de repetição **51**

5.1.1. Influenciando o fluxo de repetição 54

5.2. Estruturas de decisão **56**

6. Arrays e Blocos de Código *60*

6.1. Arrays **60**

6.1.1. Inicializando arrays 62

6.1.2. Funções de manipulação de arrays 63

6.1.3. Cópia de arrays 65

6.2. Listas de Expressões e Blocos de Código **67**

6.2.1. Premissas para utilização de Blocos de Código 67

6.2.2. Lista de expressões 68

6.2.3. Blocos de Código 70

6.2.4. Funções para manipulação de blocos de código 72

7. Funções *73*

7.1. Tipos e escopos de funções **74**

7.2. Passagem de parâmetros entre funções **77**

8. Diretivas de compilação *83*
MÓDULO 03: Desenvolvendo pequenas customizações *88*
9. ADVPL e o ERP Microsiga Protheus *88*
9.1. O Ambiente Protheus **88**
9.2. Organização e configuração inicial do ambiente Protheus **92**
9.3. O Configurador do Protheus **98**
9.3.1. Funcionalidades Abordadas 98
9.3.2. Estruturas básicas da aplicação ERP Protheus 98
9.3.3. Acessando o módulo Configurador 101
9.4. Funcionalidades do Configurador **104**
9.4.1. Dicionário de Dados da aplicação ERP 105
9.4.2. Adição de tabelas ao Dicionário de Dados 105
9.4.3. Adição de campos as tabelas do Dicionário de Dados 108
9.4.4. Adição de índices para as tabelas do Dicionário de Dados 113
9.4.5. Adição de gatilhos para os campos das tabelas do sistema 116
9.4.6. Criação de Tabelas Genéricas 118
9.4.7. Criação de Parâmetros 119
10. TOTVS Development Studio *121*
Desenvolvimento de pequenas customizações **124**
11. Acesso e manipulação de bases de dados em ADVPL *124*
11.1. Diferenças e compatibilizações entre bases de dados **125**
11.2. Funções de acesso e manipulação de dados **126**
11.3. Diferenciação entre variáveis e nomes de campos **131**
11.4. Controle de numeração seqüencial **132**
12. Customizações para a aplicação ERP *133*
12.1. Customização de campos – Dicionário de Dados **134**
12.1.1. Validações de campos e perguntas 134
12.1.2. Pictures de formação disponíveis 136
12.2. Customização de gatilhos – Configurador **138**
12.3. Customização de parâmetros – Configurador **139**
12.3.1. Funções para manipulação de parâmetros 139
12.3.2. Cuidados na utilização de um parâmetro 140
12.4. Pontos de Entrada – Conceitos, Premissas e Regras **141**
13. Interfaces visuais *142*
13.1. Sintaxe e componentes das interfaces visuais **142**
13.2. Interfaces padrões para atualizações de dados **145**
13.2.1. AxCadastro() 145
13.2.2. MBrowse() 146
13.2.3. AxFunctions() 150
APÊNDICES *152*
BOAS PRÁTICAS DE PROGRAMAÇÃO *152*
14. Utilização de Identação *152*
15. Capitulação de Palavras-Chave *153*
15.1. Palavras em maiúsculo **154**

16. Utilização da Notação Húngara 154
17. Palavras reservadas 155
LISTAS DE EXERCÍCIOS 156
Módulo 01: Introdução à programação 156
Módulo 02: A linguagem ADVPL 157
Módulo 03: Desenvolvendo pequenas customizações 160
MÓDULO 04: Desenvolvendo aplicações em ADVPL 162
18. A linguagem ADVPL 162
19. Estrutura de um Programa ADVPL 164
19.1. Áreas de um Programa ADVPL 166
20. Declaração e Atribuição de Variáveis 169
20.1. Tipo de Dados 169
20.2. Declaração de variáveis 170
20.3. Escopo de variáveis 171
20.4. Entendendo a influência do escopo das variáveis 175
21. Regras adicionais da linguagem ADVPL 176
21.1. Palavras reservadas 176
21.2. Pictures de formatação disponíveis 177
22. Programas de Atualização 178
22.1. Modelo1() ou AxCadastro() 179
22.2. Modelo2() 181
22.2.1. Componentes de uma tela no formato Modelo 2 181
22.2.2. Estrutura de um programa utilizando a Modelo2() 189
22.2.3. Função Modelo2() 197
22.3. Modelo3() 201
22.3.1. Estrutura de um programa utilizando a Modelo3() 204
22.3.2. Função Modelo3() 213
22.4. Mbrowse() 217
22.4.1. AxFunctions() 222
22.4.2. FilBrowse() 224
22.4.3. EndFilBrw() 224
22.4.4. PesqBrw() 225
22.4.5. BrwLegenda () 225
22.5. MarkBrowse() 229
23. Arquivos e Índices Temporários 233
23.1. Utilização de arquivos e índices temporários 233
23.2. Funções para manipulação de arquivos e índices temporários 233
23.2.1. CriaTrab() 233
23.2.2. dbUseArea() 234
23.2.3. IndRegua() 234
24. Relatórios não gráficos 238
24.1.1. Estrutura de relatórios baseados na SetPrint() 238
25. Manipulação de arquivos I 248

25.1. Geração e leitura de arquivos em formato texto 248
1ª. Família de funções de gravação e leitura de arquivos texto 249
2ª. Família de funções de gravação e leitura de arquivos texto 254

26. Oficina de programação I 256

26.1. Interfaces com sintaxe clássica 256

26.2. Réguas de processamento 259
26.2.1. RptStatus() 259
SETREGUA() 261
INCREGUA() 262
26.2.2. Processa() 263
SETPROC() 265
INCPROC() 265
26.2.3. MsNewProcess(). 266
26.2.4. MsAguarde(). 268
26.2.5. MsgRun() 270

26.3. ListBox() 271
26.3.1. ListBox simples 271
26.3.2. ListBox múltiplas colunas 274

26.4. ScrollBox() 276

26.5. ParamBox() 280

MÓDULO 05: Introdução a orientação à objetos 285

27. Conceitos de orientação à objetos 285

27.1. Definições 285

27.2. Conceitos Básicos 288

27.3. O Modelo de Objetos (OMT) 289
27.3.1. Objetos e Classes 289
27.3.2. Atributos 290
27.3.3. Operações e Métodos 291
27.3.4. Sugestões de desenvolvimento 292

28. Orientação a objetos em ADVPL 293

28.1. Sintaxe e operadores para orientação a objetos 293

28.2. Estrutura de uma classe de objetos em ADVPL 295

28.3. Implementação dos métodos de uma classe em ADVPL 296

MÓDULO 06: ADVPL Orientado à objetos I 300

29. Componentes da interface visual do ADVPL 300

29.1. Particularidades dos componentes visuais 307
29.1.1. Configurando as cores para os componentes 307

30. Aplicações com a interface visual do ADVPL 309

30.1. Captura de informações simples (Multi-Gets) 309
30.1.1. Enchoice() 310
30.1.2. MsMGet() 312

30.2. Captura de múltiplas informações (Multi-Lines) 314
30.2.1. MsGetDB() 315
30.2.2. MsGetDados() 319
30.2.3. MsNewGetDados() 323
30.2.3.1. Definindo cores personalizadas para o objeto MsNewGetDados() 328

30.3. Barras de botões **333**
30.3.1. EnchoiceBar() 333
30.3.2. TBar() 335
30.3.3. ButtonBar 336
30.3.4. Imagens pré-definidas para as barras de botões 339
APÊNDICES ***340***
BOAS PRÁTICAS DE PROGRAMAÇÃO ***340***
31. Arredondamento ***340***
32. Utilização de Identação ***340***
33. Capitulação de Palavras-Chave ***342***
33.1. Palavras em maiúsculo **342**
34. Utilização da Notação Húngara ***343***
35. Técnicas de programação eficiente ***343***
LISTAS DE EXERCÍCIOS ***353***
Módulo 04.1.: Desenvolvendo aplicações em ADVPL **353**
Módulo 04.2.: Oficina de Programação I **358**
Módulo 05: Introdução a orientação à objetos **360**
Módulo 06: ADVPL orientado à objetos I **361**
Projeto: Avaliação prática do treinamento de ADVPL Intermediário **363**
MÓDULO 07: ADVPL Orientado à objetos II ***364***
36. Componentes da interface visual do ADVPL ***364***
TSRVOBJECT() 366
TFONT() 369
MSDIALOG() 370
TDIALOG() 371
TWINDOW() 374
TCONTROL() 377
BRGETDDB() 378
MSCALEND() 381
MSCALENDGRID() 383
MSSELBR() 385
MSWORKTIME() 388
SBUTTON() 391
TBAR() 393
TBITMAP() 395
TBROWSEBUTTON() 397
TBTNBMP() 399
TBTNBMP2() 401
TBUTTON() 403
TCBROWSE() 405
TCHECKBOX() 409
TCOLORTRIANGLE() 411
TCOMBOBOX() 413
TFOLDER() 416
TGET() 418
TGROUP() 420
THBUTTON() 422
TIBROWSER() 423
TLISTBOX() 425
TMENU() 429

TMENUBAR() 430
TMETER() 431
TMSGRAPHIC() 434
TMSGBAR() 440
TMSGITEM() 442
TMULTIBTN() 443
TMULTIGET() 446
TOLECONTAINER() 448
TPAGEVIEW() 450
TPANEL() 452
TRADMENU() 453
TSBROWSE() 457
TSAY() 461
TSCROLLBOX() 463
TSIMPLEEDITOR() 464
TSLIDER() 469
TSPLITTER() 472
TTABS() 476
TTOOLBOX() 478
TWBROWSE() 481
VCBROWSE() 485

36.1. Particularidades dos componentes visuais 491
36.1.1. Configurando as cores para os componentes 491

37. Aplicações com a interface visual do ADVPL 493

37.1. Captura de informações simples (Multi-Gets) 493
37.1.1. Enchoice() 494
37.1.2. MsMGet() 496

37.2. Captura de múltiplas informações (Multi-Lines) 498
37.2.1. MsGetDB() 499
37.2.2. MsGetDados() 503
37.2.3. MsNewGetDados() 507
37.2.3.1. Definindo cores personalizadas para o objeto MsNewGetDados() 512

37.3. Barras de botões 517
37.3.1. EnchoiceBar() 517
37.3.2. TBar() 519
37.3.3. ButtonBar 520
37.3.4. Imagens pré-definidas para as barras de botões 523

38. Outras aplicações da interface visual do ADVPL 524

38.1. MaWndBrowse() 524
38.1.1. Enchoice para Arquivos Temporários 527

38.2. DbTree() 537

38.3. MsSelect() 542

39. Introdução à relatórios gráficos 545

39.1. TReport() 545
39.1.1. Introdução 545
Finalidade 545
Descrição 546
Pré-Requisitos 546
39.1.2. Impressão do relatório personalizável 547
39.1.2.1. Parâmetros de impressão 547
Impressão 547
Arquivo 547
Spool 548

E-mail....548
Papel....549
Tamanho do papel....549
Formato da impressão....549
Configurações....549
Título....549
Ordem....549
Layout....549
Preview....549
Executar em segundo plano....550
39.1.3. Personalização....550
39.1.3.1. Editando o layout do relatório....550
Nova estrutura do relatório TReport:....551
39.1.4. Definindo a Função ReportDef()....551
DEFINE REPORT....552
DEFINE SECTION....552
DEFINE CELL....552

MÓDULO 08: Aplicações ADVPL para o ERP....553

40. Protheus e o TOPCONNECT / TOTVS DbAccess....553

40.1. Características técnicas com o uso do TOTVS DbAccess....554
Comportamento de Queries – Colunas Calculadas....554
Comportamento diferenciado com Bandos de Dados PostGres....556
Conceito de Índices Permanentes e Diferenças das RDDs....557
Funcionamento Interno....557
Quebra de Compatibilidade com CodeBase/DBF....558
Lista dos códigos de erro do TOPConnect / DbAccess....559

40.2. Funções ADVPL para TOPCONNECT / TOTVS DbAccess....560
Lista das funções de interação com a aplicação TopConnect / DbAccess:....560
Lista das funções acessórias utilizadas nos fontes como facilitadoras:....560
Funções de interação com o TopConnect / DbAccess....561
TCCANOPEN ()....561
TCCONTYPE()....561
TCDELFILE()....562
TCGENQRY()....563
TCGETDB()....563
TCLINK()....564
TCQUERY()....566
TCQUIT()....567
TCSETCONN()....567
TCSETFIELD()....568
TCSPEXEC()....570
TCSPEXIST()....572
TCSQLERROR()....572
TCSQLEXEC()....573
TCSRVTYPE()....573
TCUNLINK()....574
TCCHKOBJ()....575
TCEXEERROR()....575
TCPGMEXE()....576
TCSYSEXE()....576
Funções acessórias para uso em fontes com interação com o TOTVS DbAccess....577
CHANGEQUERY()....577
RETFULLNAME()....578
RETSQLCOND()....579
RETSQLNAME()....579
RETSQLTABLE()....580
SQLCOPY()....580

SQLORDER() ... 581
SQLTOTRB() ... 582

40.3. Aplicações com controle de comunição com o Banco de Dados ... 583
40.3.1. MaWndBrowse com Alias Temporário gerado por Query ... 583
Exemplo: MaWndBrowse com Alias Temporário gerado por Query ... 584
40.3.2. Banco de dados de interface ... 595
Considerações relevantes sobre as funções TCLink() e TCSetConn() ... 595
Considerações complementares sobre o conceito de Banco de Dados de Interface ... 596

40.4. Embedded SQL – Facilitador de Query's ... 598
Disponibilidade do Recurso ... 598
Características operacionais - Sintaxe ... 599
Limitação: ... 599
Erros de Compilação ... 600
Erros de Execução ... 600
Características operacionais - depuração ... 601
Função auxiliar - GETLastQuery() ... 601

41. Funcionalidade MsExecAuto ... 603
Sobre a MsExecAuto e Rotinas Automáticas ... 603
Quando utilizar a funcionalidade MsExecAuto ? ... 604
Processos da aplicação ERP com tratamentos para execução por MsExecAuto ... 604
Quando não utilizar a funcionalidade MsExecAuto ... 605

41.1. Desenvolvendo aplicações com MsExecAuto ... 606
Estrutura de uma rotina com execução de MsExecAuto ... 606
Variáveis de controle ... 606
Montagem dos arrays de parâmetros ... 606
Definição dos parâmetros específicos da rotina que será executada ... 610
Controle de transação ... 611
Tratamento de mensagens de erro ... 613
Aplicação completa de importação utilizando MsExecAuto: Carga de imobilizado ... 616

42. Recursos de envio de e-mail ... 629
Funções para manipulação de e-mails ... 629
Detalhamento das funções de manipulação de e-mails ... 629
CALLPROC() ... 629
MAILSMTPON() ... 630
MAILPOPON() ... 630
MAILSMTPOFF() ... 631
MAILPOPOFF() ... 631
MAILRECEIVE() ... 632
MAILAUTH() ... 633
POPMSGCOUNT() ... 634
MAILSEND() ... 634
MAILGETERR() ... 635
Exemplos de utilização das funcionalidades de envio e recebimento de e-mail ... 635
Envio de mensagens utilizando sintaxe clássica ... 635
Envio de mensagens utilizando funções ... 638
Recebimento de mensagens utilizando funções ... 641

43. Integração básica com MsOffice ... 643

43.1. Exportação para EXCEL ... 643
DLGTOEXCEL() ... 643
Exemplo de exportação para o Microsoft Excel utilizando a função DlgToExcel() ... 644

APÊNDICES ... 646

Relação de imagens para aplicações visuais ... 646

LISTAS DE EXERCÍCIOS ... 654

Projeto: Avaliação prática do treinamento de ADVPL Avançado **659**
REFERÊNCIAS BIBLIOGRÁFICAS ***660***

MÓDULO 01: Introdução à programação

1. Lógica de Programação e Algoritmos

No aprendizado de qualquer linguagem de programação é essencial desenvolver os conceitos relacionados a lógica e a técnica da escrita de um programa.

Com foco nesta necessidade, este tópico irá descrever resumidamente os conceitos envolvidos no processo de desenvolvimento de um programa através dos conceitos relacionados à:

- ☑ 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 seqüência lógica para o desenvolvimento. Então o que é lógica?

Lógica de programação é a técnica de encadear pensamentos para atingir determinado objetivo.

Seqüência Lógica

Estes pensamentos, podem ser descritos como uma seqüência de instruções, que devem ser seguidas para se cumprir uma determinada tarefa.

Seqüência Lógica são passos executados até atingir um objetivo ou solução de um problema.

Instruções

Na linguagem comum, entende-se por instruções "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 têm que ser executadas em uma ordem adequada – não se pode descascar as batatas depois de fritá-las.

Dessa maneira, uma instrução tomada em separado não tem muito sentido; para obtermos o resultado, precisamos colocar em prática o conjunto de todas as instruções, na ordem correta.

Instruções são 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 seqüência finita de passos que levam a execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma seqüência 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 podemos citar os algoritmos das operações básicas (adição, multiplicação, divisão e subtração) de números reais decimais. Outros exemplos seriam os manuais de aparelhos eletrônicos, como um videocassete, que explicam passo-a-passo como, por exemplo, gravar um evento.

Até mesmo as coisas mais simples, podem ser descritas por seqüências 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 a 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 construção do Algoritmo

Para escrever um algoritmo precisamos descrever a seqüência 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 sentido dúbio.

Fases

Para implementar de um algoritmo de simples interpretação e codificação é necessário inicialmente dividir o problema apresentado em três fases fundamentais, as quais são:

- ☑ **ENTRADA**: São os dados de entrada do algoritmo;
- ☑ **PROCESSAMENTO**: São os procedimentos utilizados para chegar ao resultado final;
- ☑ **SAÍDA**: São os dados já processados.

1.2.1. Estudando algoritmos

Neste tópico serão demonstrados alguns algoritmos do cotidiano, os quais 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 ganho

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 lâmpada queimada
5. Colocar lâmpada nova

Descascar batatas

1. Pegar faca, bacia e batatas
2. Colocar água na bacia
3. Enquanto houver batatas, descascar 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 centro estiver livre: jogue no centro
   1.3. Senão, se o adversário possuir 2 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

Após desenvolver um algoritmo ele deverá sempre ser testado. Este teste é chamado de ***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
3	8.0
4	8.0
5	7.0
6	7.0

3. Somar todas as notas: 45
4. Dividir a soma das notas, pelo total de notas informado: 45/6 → 7.5
5. Exibir a média obtida: Mensagem(Média: 7.5)

Anotações

2. Estruturas de programação

2.1. Diagrama de bloco

O diagrama de blocos é uma forma padronizada e eficaz para representar os passos lógicos de um determinado processamento.

Com o diagrama podemos definir uma seqüência 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.

Importante

Cada símbolo irá conter uma descrição pertinente a forma com o qual o mesmo foi utilizado no fluxo, indicando o processamento ou a informação que o mesmo representa.

Algoritmo 01: 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

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 a situações que influenciam na execução e 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

2.2.1. Estruturas de decisão

Os comandos de decisão são utilizados em algoritmos cuja solução não é obtida através da utilização de ações meramente seqüenciais, permitindo que este avalie 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 a análise da condição resultar em um valor verdadeiro.

Representação 02: IF...ELSE somente com ação para situação verdadeira

Importante

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 seqüência, 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 a primeira condição verdadeira será sua ação vinculada 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.

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

Nesta estrutura, o conjunto de ações será 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

Existem diversas variações para a estrutura WHILE...END, na qual há 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

Nesta estrutura, o conjunto de ações será executado uma quantidade de vezes definida, normalmente referenciada como “passo”.

Para cada “passo” realizado pela estrutura FOR...TO...NEXT, será avaliada uma condição que verificará se foi atingido o número de execuções previamente definido. Desta forma a estrutura compreende um controle de número de “passos” executados, o qual é 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

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, o qual 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.

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 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 realizem 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 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. 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.

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 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, 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 as 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 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 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. 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 onde 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
```

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 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 a **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 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 necessário mais se referenciar ao seu conteúdo, e sim ao seu nome.

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.

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 deve-se 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 nNumero := 10
```

Esta linha de código declara uma variável chamada nNumero indicando que 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 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 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 mantêm seu valor atual quando a execução da função Pai termina. Entretanto, somente pode ser acessada por execuções subseqüentes 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, 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.

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 caracter "_" 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 *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.)
```

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. 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).

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 caracter "_" antes do nome de variáveis PRIVATE e PUBLIC. Maiores informações avaliar 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 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.

4.5. Operações com Variáveis

4.5.1. Atribuição de 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, pode-se mudar o tipo da 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)
```

Neste 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:

:=	Atribuição Simples
+=	Adição e Atribuição em Linha
-=	Subtração e Atribuição em Linha
***=**	Multiplicação e Atribuição em Linha
/=	Divisão e Atribuição em Linha
****= ou ^=**	Exponenciação e Atribuição em Linha
%=	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 pode-se 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, pode-se 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. Com eles pode-se economizar digitação:

Operador	Exemplo	Equivalente a
+=	X += Y	X = X + Y
-=	X -= Y	X = X - Y
*=	X *= Y	X = X * Y
/=	X /= Y	X = X / Y
**= ou ^=	X **= Y	X = X ** Y
%=	X %= Y	X = X % Y

Operadores de Incremento/Decremento

A linguagem ADVPL possui operadores para realizar incremento ou decremento de variáveis. Entende-se por incremento aumentar o valor de uma variável numérica em 1 e entende-se por decremento diminuir o valor da variável em 1. Os operadores são:

++	Incremento Pós ou Pré-fixado
--	Decremento Pós ou Pré-fixado

Os operadores de decremento/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 destas 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 seqüência. 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 intero 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. Pode-se 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 pode-se substituir Y na terceira linha:

```
03 B := &"X + 1"
O operador de macro cancela as aspas:
03 B := X + 1
```

Pode-se perceber que o operador de macro remove as aspas, o que deixa um pedaço de 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 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 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 caracter 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 caracter, sendo o resultado no formato “DD/MM/AAAA”.

DTOS()

Sintaxe	**DTOS(dData)**
Descrição	Realiza a conversão de uma informação do tipo data em um caracter, sendo o resultado no formato “AAAAMMDD”.

STOD()

Sintaxe	**STOD(sData)**
Descrição	Realiza a conversão de uma informação do tipo caracter 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 caracter 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	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").

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 caracter 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 caracter 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 a 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, a qual não foi definida na função em execução.

VALTYPE()

Sintaxe	**VALTYPE(cVariável)**
Descrição	Determina o tipo do conteúdo de uma variável, a qual foi definida na função em execução.

Anotações

5. Estruturas básicas de programação

O ADVPL suporta várias estruturas de controle que permitem mudar a seqüência de fluxo de execução de um programa. Estas estruturas permitem a execução de código baseado em condições lógica e a repetição da execução de pedaços de código 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, pode-se desejar imprimi-lo quatro vezes. Claro, pode-se simplesmente chamar a função de impressão quatro vezes em seqüência, 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 **variável privada**.
nValorInicial TO nValorFinal	nValorInicial é o valor inicial para o contador; nValorFinal é o valor final para o contador. Pode-se utilizar valores numéricos literais, variáveis ou expressões, contanto que o resultado seja do tipo de 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. Pode-se utilizar valores numéricos literais, variáveis ou expressões, contanto que o resultado seja do tipo de dado numérico.
Comandos	Especifica um 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. Pode-se colocar 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. Pode-se colocar o comando LOOP em qualquer lugar entre o FOR e o NEXT.

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 (Variavel) 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
```

Este 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 um 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. Pode-se colocar 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 continuarão sendo executados.

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.

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
```

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. Desta 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

// Caso o produto esteja 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. Desta forma, todas as operações que seriam realizadas dentro da estrutura de repetição após o EXIT serão desconsideradas, e o programa irá continuar 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

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 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 são ignorados. Neste 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.

Pode-se 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.). Caso a lExpressaoX avaliada resulte em falso (.F.) será avaliada a próxima expressão lExpressaoX vinculada ao próximo comando ELSEIF, ou caso o mesmo não exista será executada a ação definida parao comando ELSE.
Comandos	Conjunto de comandos ADVPL que serão executados dependendo da avaliação da expressão lógica em lExpressao.

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...**	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 seja encontrada. Ao terminar de executar esse conjunto de comandos, a execução continua com o primeiro comando seguinte ao ENDCASE. Se uma expressão CASE resultar em falso (.F.), o conjunto de comandos seguinte a esta até a próxima cláusula é ignorado. 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).
OTHERWISE **Comandos**	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.

Importante

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
```

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 um pedaço diferente de 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, pode-se agrupá-las em um outro array, do mesmo modo que muitos registros são agrupados em uma tabela de banco de dados:

```
aFuncts := {aFunct1, aFunct2, aFunct3}
```

Que é equivalente a isso:

```
aFuncts := { {"Pedro"  , 32, .T.}, ;
           {"Maria"  , 22, .T.}, ;
           {"Antônio", 42, .F.} }
```

aFuncts 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(aFuncts)
Alert(aFuncts[nCount, FUNCT_NOME])
// O acesso a elementos de um array multidimensional
// pode ser realizado também desta forma:
// aFuncts[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 criada 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 de 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 podem 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 permite que as informações sejam armazenadas e organizadas das mais diversas formas.

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 o obtido na execução da "cópia".

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 represente 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.

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 onde as informações estão realmente armazenadas.

Como foi copiado o "mapa" e não as informações, qualquer ação utilizando o rótulo aAlunos irá afetar 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 deseja-se obter um array com a mesma estrutura e informações que compõe outro array já existente.

6.2. Listas de Expressões e Blocos de Código

Blocos de código são 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.

6.2.1. Premissas para utilização de Blocos de Código

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))
```

Pode-se escrever:

```
// 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 desta 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 isso acontece ao contrário, 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. Pode-se dizer que o zero foi "propagado através da expressão".

Segunda premissa

Em ADVPL pode-se juntar diversas linhas de código em uma única linha física de comando. Por exemplo, o código:

```
If lAchou
Alert("Cliente encontrado!")
Endif
pode ser escrito assim:
If lAchou ; Alert("Cliente encontrado!") ;
Endif
```

O ponto-e-vírgula indica ao ADVPL que a nova linha de código está para começar. Pode-se 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, conseqüentemente, a 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 sua avaliação (seja para atribuir em uma variável, exibir para o usuário ou imprimir em um relatório), que será impresso 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 linha de código em uma , utilizando 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).

Neste 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 de expressões. 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 pode-se 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 que uma expressão simples pode ser utilizada, pode-se utilizar uma lista de expressões. E ainda, pode-se fazer com que várias coisas aconteçam onde normalmente apenas uma aconteceria.

```
X := 10 ; Y := 20
If X > Y
Alert("X")
Z := 1
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 pequenos pedaços de código, como se fosse um pequeno programa, utilizados para diminuir 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 avalias todas as expressões contidas neste bloco de código, retornando o resultado da última expressão avaliada.

Passando Parâmetros

Já que 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 eventos de objetos são acionados ou para modificar o comportamento padrão de algumas funções.

Por 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 está 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 irá então trocar os valores de lugar e seguir 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, pode-se 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, pode-se utilizar 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 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.

DBEVAL()

Sintaxe	**Array(bBloco, bFor, bWhile)**
Descrição	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.

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. Funções

A maior parte das rotinas que queremos escrever em programas são compostas de um conjunto de comandos, rotinas estas que 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 pode-se passar parâmetros, o quais contém os dados e 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, o que permite executar uma função escrevendo:

```
Calcula(parA, parB, parC) // Chamada da função em uma rotina
```

E a função estar escrita:

```
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 à 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.

Existe um ditado que diz que:

"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 DEM (Documentação Eletrônica Microsiga) 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 do menu são funções, sendo que em um repositório não podem 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 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”.

1. Funções do tipo Function() somente podem ser executadas através dos módulos do ERP.
2. Somente poderão ser compiladas funções do tipo Function() se o MP-IDE possuir uma autorização especial fornecida pela Microsiga.
3. 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”.

1. 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 para evitar resultados indesejados durante a execução da aplicação.

2. 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 DEM o tópico sobre preparação de ambiente e a documentação sobre a função RpcSetEnv().

Static Function()

Funções ADVPL tradicionais, cuja visibilidade está restrita as 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 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 de função principal / aplicação.

2. Redefinir uma função padrão da aplicação, adequando-a as necessidades específicas de uma função principal / aplicação.

3. Proteger 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 código 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 a Inteligência Protheus da Microsiga.

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 sua execução a variável nFator.

Com base nesta função podemos descrever duas forma 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. Neste 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 a variável definida como primeiro parâmetro da função chamado, 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.

Duas características da linguagem ADVPL tornam necessária uma atenção especial na chamada de funções:

1. 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 através de uma atribuição.

2. Os parâmetros de uma função são atribuídos de acordo com a ordem com os quais são definidos na chamada da mesma. Não é realizada nenhum tipo de consistência em relação aos tipos dos conteúdos e obrigatoriedade de parâmetros nesta ação.

Os parâmetros de uma função são caracterizados como variáveis de escopo LOCAL para efeito de execução.

Desta forma os mesmos não devem ser definidos novamente como LOCAL na área de definição e inicialização de variáveis, pois caso isto ocorra haverá a perda dos valores recebidos pela redefinição das variáveis na função.

Caso seja necessário garantir um conteúdo padrão para um determinado parâmetro deverá ser utilizado o identificador DEFAULT conforme detalhado 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. Neste caso foi informada a variável nFatorUser, a qual será definida pelo usuário através 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 a variável definida como primeiro parâmetro da função chamado, 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 := 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.

Desta 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 a qual permite que variáveis de escopo LOCAL tenham seu conteúdo manipulado por funções específicas, mantendo o controle destas variáveis restrito a função que as definiu e as 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.

```
FuncaoA()
xParamY := 10

Funcao(xParamY)

Alert(cValToChar(xParamY)
Return
```

```
FuncaoB(xParam1)

xParam1 := 15

Return
```

Passagem de parâmetros tradicional – Duas variáveis x Duas áreas de memória

```
FuncaoA()
xParamY := 10

Funcao(xParamY)

Alert(cValToChar(xParamY)
Return
```

```
FuncaoB(xParam1)

xParam1 := 15

Return
```

Passagem de parâmetros por referência – Duas variáveis x uma única área de memória

Anotações

Desta forma a função chamada tem acesso não apenas ao conteúdo, mas a variável em si, pois a área de memória é a variável, e qualquer alteração nesta será visível a função chamadora quando tiver o retorno da função chamadora.

Tratamento de conteúdos padrões para parâmetros de funções

O tratamento de conteúdos padrões para parâmetros de funções é muito utilizado nas funções padrões da aplicação ERP, de forma a garantir a correta execução destas funções por qualquer função chamadora, evitando situações de ocorrências de erros pela falta da definição de parâmetros necessários a correta utilização da função.

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.

Desta forma o uso do identificador DEFAULT permite ao desenvolvedor garantir que na utilização da função determinados parâmetros terão o valor com um tipo adequado a 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, caso o parâmetro nFator não seja informado na função chamadora, o mesmo terá seu conteúdo definido como 1.

Se este 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, portando 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 mismath on +=, expected N → U" pois os tipos das variáveis envolvidos na operação são diferentes: nCnt → nulo (U) e 1 → numérico (N).

Dica

Caso o parâmetro que possui a opção DEFAULT descrita no fonte seja informado, a linha de DEFAULT não será executada, mantendo desta forma o conteúdo passado pela função chamadora.

Anotações

8. Diretivas de compilação

O compilador ADVPL possui uma funcionalidade denominada pré-processador, o qual nada mais é do que 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, as 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**

Lembre-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, os quais devem ser utilizados segundo a aplicação que será desenvolvida, o que permitirá a utilização de recursos adicionais definidos para a linguagem, implementados pela área de Tecnologia da Microsiga.

Os includes mais utilizados nas aplicações ADVPL desenvolvidas para o ERP são:

- ☑ **PROTHEUS.CH**: diretivas de compilação 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 utilizadas pela linguagem ADVPL que complementam esta funcionalidade de compatibilidade com a sintaxe Clipper, tais como:

- **DIALOG.CH**
- **FONT.CH**
- **INI.CH**
- **PTMENU.CH**
- **PRINT.CH**

Fique atento

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 – DbAcess):

- TCQUERY

☑ **TBICONN.CH:** Permite a utilização da sintaxe tradicional na definição de conexões com a aplicação Server do ambiente ERP, através da seguintes sintaxes:

- CREATE RPCCONN
- CLOSE RPCCONN
- 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 (DbAcess), 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.

Caso o diretório de includes não esteja 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 variáveis diversas utilizadas pela aplicação em diferentes idiomas. Desta forma é normal verificar que um fonte possui um arquivo ".CH" com o mesmo nome, o que caracteriza este tipo de include.

Diretiva: #DEFINE

A diretiva #DEFINE permite que o desenvolvedor crie novos termos para serem utilizadas 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.

Desta 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 contém 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 esta 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.

Através destas 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.

Desta forma, 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, o qual irá gerar um código compilado de acordo com a análise dos parâmetros de ambiente.

Estas diretivas de compilação estão normalmente associadas as 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
```

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.

A aplicação ERP disponibiliza a variável de escopo PUBLIC - __LANGUAGE, a qual contém uma string que identifica o idioma em uso pelo sistema, cujo os 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 formado 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
```

Os bancos de dados padrão AS400 não permitem a execução de queries no formato SQLANSI através da ferramenta TOPCONNECT (DbAcess).

Desta 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 estes 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.

Esta diretiva permite que o desenvolvedor defina para o compilador como uma expressão deverá 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.

MÓDULO 03: Desenvolvendo pequenas customizações

9. ADVPL e o ERP Microsiga Protheus

O ADVPL (Advanced Protheus Language) é uma linguagem de programação desenvolvida pela Microsiga e que contém todas as instruções e funções necessárias ao desenvolvimento de um sistema, independente de sua complexidade.

O PROTHEUS, por outro lado, é uma plataforma tecnológica que engloba um Servidor de Aplicação, um Dicionário de Dados e as Interfaces para conexão com o usuário. É o Protheus que executa o código ADVPL e o devido acesso à base da dados.

O Protheus é composto pelo ERP (que engloba, além das funcionalidades descritas nos capítulos anteriores, mais de trinta verticais aplicadas a 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 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, o qual 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 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, o 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). Neste 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 (TOTVSAPPSERVER.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.
- **APPSERVER:** 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 (TOTVSAPPSERVER.INI e TOTVSSMARTCLIENT.INI) existentes nas respectivas pastas APPSERVER e SMARTCLIENT.

Os parâmetros do TOTVSAPPSERVER.INI são lidos pelo programa TOTVSAPPSERVER.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 APPSERVER e TOTVS SMARTCLIENT.

Figura: Links dos parâmetros de configuração

Para que o TOTVS AppServer e o TOTVS SmartClient sejam executados, os arquivos TOTVSAPPSERVER.INI e TOTVSSMARTCLIENT.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

Destino: c:\protheus\bin\appserver\totvsappserver.exe - console

Iniciar em: c:\protheus\bin\appserver

☑ **-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, pode-se executá-lo com a opção de Linha de Comando.

Destino: c:\protheus\bin\smartclient\totvssmartcliente.exe –M

Iniciar em: c:\protheus\bin\smartclient

☑ **-Q (Quiet)**

Indica que o TOTVS Smart Client, não deverá mostrar o Splash (Imagem 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 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, à linguagem do país em que está sendo utilizado e as máscaras de edição e formatação.

[ENVIRONMENT]
SOURCEPATHC:\PROTHEUS\APO
ROOTPATH= C:\MP811\PROTHEUS_DATA
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 (TOTVSAPPSERVER.INI).

Já o arquivo de parâmetros do Protheus Remote (TOTVSSMARTCLIENT.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=SIGACFG
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 TOTVSAPPSERVER.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

9.3.1. Funcionalidades Abordadas

O objetivo deste tópico não é abranger todo 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
  - o Tabelas do sistema
  - o Validações de campos
  - o Índices de tabelas
  - o 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 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	Seqüência de documentos (+1)
SXF	Seqüência 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, POSTGREE, etc.).

No caso do SQL, o acesso é feito através do TOPCONNECT / DBACESS, 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 especificas 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
SIGALOJ	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, os quais 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 seqüencial 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).
S	**B**	Cadastros 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**	Cadastros e movimentações do ambiente Financeiro.
S	**F**	Cadastros e movimentações Fiscais (Cabeçalhos das notas fiscais de entrada e saída, cadastro de tipos de entrada e saída, livros fiscais, entre outros).
S	**G**	Cadastros do ambiente de Planejamento e Controle de Produção
S	**H**	Movimentos do ambiente de Planejamento e Controle de Produção
S	**I**	Cadastros e movimentos do ambiente Contábil (descontinuado)
S	**N**	Cadastros e movimentos do ambiente Ativo Fixo
S	**R**	Cadastros 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
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 a família SZ.

Índices

Cada tabela do sistema possui seus índices definidos no arquivo de configuração SIX, o qual pode ser atualizado através 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 bancos de dados ISAM, será 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 seqüencial em função do nome da tabela original.

As especificações das chaves de índices de cada um 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

Anotações

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 interpretadas 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.

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

4. 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 número 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.

> 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, nas versões com banco de dados relacional não são utilizadas. Essa pasta será criada dentro da pasta indicada na configuração do sistema como ROOTTPATH.

- ☑ 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 as 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, 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 Campos ( ), para que sejam exibidos os campos disponíveis para a tabela no arquivo de sistema SX3.

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 (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 (OK).

6. Confirme as atualizações para a tabela selecionada com o botão Confirmar (Confirmar).

7. Atualize as configurações do sistema com o botão Atualizar (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 ( ). 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 as dados diretamente em bancos de dados de ambientes TOPCONNECT (DbAcess).

9.4.5. Adição de gatilhos para os campos das tabelas do sistema

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 cadastros 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 sitema 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 Seqüência.
- ☑ 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.

Figura: Conjunto de Tabelas já cadastras 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 botão Confirmar (

).

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 cadastras 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 uma 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 executa-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 irá executá-la;
- ☑ 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 seqüência 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.

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 genéricas
- ☑ Funções de manipulação de dados específicas para ambientes TOPCONNECT / DBACCESS

Funções de manipulação de dados genéricas

As funções de manipulação de dados ditas como genéricas 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 utilizadas 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 (DbAcess).

A integração entre a aplicação ERP e a ferramenta TOPCONNECT permite que as funções de acesso e manipulação de dados escritas em ADVPL sejam interpretadas e convertidas para uma sintaxe compatível com o padrão SQL ANSI e desta forma aplicadas 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 ODBCs.

As funções específicas para ambientes TOPCONNECT serão abordas 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 irá optar 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 seqüencial tendo como base o nome da tabela ao 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 destas operações.

Deve ser considerada a possibilidade de utilização de índices temporários para processos específicos, os quais serão criados em tempo de execução da rotina. Este 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 seqüencial e ascendente que funciona como o endereço base daquela informação.

Este ID, mas conhecido como RECNO ou RECNUMBER é gerado no momento de inclusão do registro na tabela e somente será alterado caso a estrutura dos dados desta tabela sofra alguma manutenção. Dentre as manutenções que uma tabela de dados ISAM pode sofrer pode-se citar a utilização do comando PACK, o qual irá apagar fisicamente os registros deletados da tabela forçando uma renumeração dos identificadores de todos os registros. Esta situação também torna necessária a recriação de todos os índices vinculados àquela tabela.

Isto 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 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 através da criação de campos de controle específicos: R_E_C_N_O_, D_E_L_E_T_ e R_E_C_D_E_L.

Estes 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 seqüencial ao banco de dados.

O campo D_E_L_E_T_ é tratado internamente pela aplicação ERP como um "flag" ou marca de exclusão. Desta 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 nas 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 as informações (registros e campos).

Dentre as funções ADVPL disponíveis para acesso e manipulação de informações, este material irá detalhar as seguintes opções:

- **SELECT()**
- **DBSELECTAREA()**
- **DBSETORDER()**
- **DBSEEK() E MSSEEK()**
- **DBSKIP()**
- **DBGOTO()**
- **DBGOTOP()**
- **DBGOBOTTON()**
- **DBSETFILTER()**
- **RECLOCK()**
- **SOFTLOCK()**
- **MSUNLOCK()**
- **DBDELETE()**
- **DBUSEAREA()**
- **DBCLOSEAREA()**

DBRLOCK()

Sintaxe	DBRLOCK(xIdentificador)
Descrição	Função de base de dados, que efetua o lock (travamento) do registro identificado pelo parâmetro xIdentificador. Este parâmetro pode ser o Recno() para tabelas em formado ISAM, ou a chave primária para bancos de dados relacionais. Se o parâmetro xIdentificador 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.

DBCLOSEAREA()

Sintaxe	DbCloseArea()
Descrição	Permite que um alias presente na conexão seja fechado, o que viabiliza seu reuso em outro operação. Este comando tem efeito apenas no alias ativo na conexão, sendo necessária sua utilização em conjunto com o comando DbSelectArea().

DBCOMMIT()

Sintaxe	DBCOMMIT()
Descrição	Efetua todas as atualizações pendentes na área de trabalho ativa.

DBCOMMITALL()

Sintaxe	DBCOMMITALL()
Descrição	Efetua todas as atualizações pendentes em todas as área de trabalho em uso pela thread (conexão) ativa.

DBDELETE()

Sintaxe	DbDelete()
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 uma busca (seek) prévio.

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	**DbSeek(cChave, lSoftSeek, lLast)**
Descrição	**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.

DBSKIP()

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 com sendo a área ativa. Todas as operações subseqüentes que fizerem referência a uma área de trabalho a 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 utilizada 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 seja 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.

DBUNLOCKALL()

Sintaxe	**DBUNLOCKALL()**
Descrição	Libera o travamento de todos os registros de todas as áreas de trabalho disponíveis na thread (conexão) ativa.

DBUSEAREA()

Sintaxe	**DbUseArea(lNovo, cDriver, cArquivo, cAlias, lComparilhado,; lSoLeitura)**
Descrição	Define um arquivo de base de dados como uma área de trabalho disponível na aplicação.

MSUNLOCK()

Sintaxe	**MsUnLock()**
Descrição	Libera o travamento (lock) do registro posicionado confirmando as atualizações efetuadas neste registro.

RECLOCK()

Sintaxe	**RecLock(cAlias,lInclui)**
Descrição	Efetua o travamento do registro posicionado na área de trabalho ativa, permitindo a inclusão ou alteração das informações do mesmo.

RLOCK()

Sintaxe	**RLOCK() → lSucesso**
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	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.

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 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
```

Esta linha de comando identifica que o valor atribuído à variável cRes deve ser o valor da variável de memória chamada NOME.

```
cRes := FIELD->NOME
```

Neste 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 aberto, para evitar a necessidade de selecionar a área antes de acessar o conteúdo de terminado 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 desta tabela.

```
cRes := SA1->NOME // SA1 – Cadastro de Clientes
```

Para maiores detalhes sobre abertura de arquivos com atribuição de apelidos, consulte a documentação sobre acesso a banco de dados ou a documentação da função dbUseArea().

Os alias das tabelas da aplicação ERP são padronizados em três letras, que correspondem as iniciais da tabela. As configurações de cada ALIAS utilizado pelo sistema podem ser visualizadas através do módulo Configurador -> Bases de Dados -> Dicionários -> Bases de Dados.

11.4. Controle de numeração seqüencial

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.

Os campos que recebem o tratamento de numeração seqüencial pela aplicação ERP não devem ser considerados como chave primária das tabelas aos quais 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 seqüência 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 ser disponível mesmo que naquele momento números maiores já tenham sido oferecidos e utilizados.

Com isso, mesmo que tenhamos vários processos solicitando numerações seqüenciais para uma mesma tabela, como por exemplo inclusões simultâneas de pedidos de vendas, teremos para cada pedido um número exclusivos e sem o intervalos e numerações não utilizadas.

Funções de controle de semáforos e numeração seqüencial

A linguagem ADVPL permite a utilização das seguintes funções para o controle das numerações seqüenciais utilizadas nas tabelas da aplicação ERP:

- **GETSXENUM()**
- **CONFIRMSXE()**
- **ROLLBACKSXE()**

GETSXENUM()

Sintaxe	**GETSXENUM(cAlias, cCampo, cAliasSXE, nOrdem)**
Descrição	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.

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 outras conexões.

12. Customizações para a aplicação ERP

Neste tópico serão abordas 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 nesta premissa, a utilização de validações no Dicionário de Dados (SX3) ou nas Perguntas de Processos e Relatórios (SX1) deverá focar sempre na utilização de funções ou expressões que resultem em um retorno lógico.

Através do módulo Configurador é possível alterar as propriedades de um campo ou de uma pergunta de forma a incluir 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	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.

EXISTCPO()

Sintaxe	**ExistCpo(cAlias, cConteudo, nIndice)**
Descrição	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.

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 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 DEM – Documentação Eletrônica Microsiga, 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 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.

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 inicias 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 digitação de informações. As funções que podem ser utilizadas no gatilho estão diretamente relacionadas a 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:

  - 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 a informações de tabelas de movimentos 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 destes 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 este campo que determina qual 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 do parâmetro: de forma similar a uma variável, um parâmetro terá um tipo de conteúdo pré-definido em seu cadastro.
  Esta informação é utilizada pelas funções da aplicação ERP na interpretação do conteúdo do parâmetro e retorno desta informação a 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. Estes parâmetros macro executáveis tem como única característica em comum seu tipo: caractere, mas não existe nenhum identificador explicito que permite a fácil visualização de quais parâmetros possuem um retorno simples e de quais parâmetros terão 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, no qual 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 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. Caso o parâmetro não exista será exibido um help do sistema informando a ocorrência.

GETNEWPAR()

Sintaxe	**GETNEWPAR(cParametro, cPadrao, cFilial)**
Descrição	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 conseqüência não será exibida a mensagem de help.

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	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.

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 permitam a configuração de expressões, e por conseqüência a utilização de funções para definir o retorno que será obtido com a consulta deste parâmetro, ***é expressamente proibido o uso de funções em parâmetros para manipular informações da base de dados do sistema.***

Caso haja a 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 desta restrição é simples:

- ☑ As rotinas da aplicação ERP não protegem a consulta de conteúdos de parâmetros quanto a gravações realizadas dentro ou fora de uma transação.
- ☑ Desta forma, quaisquer 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 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 especificas 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 o mesmo 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 sua situação imediatamente anterior à execução do ponto restaurada 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 deste um conteúdo simples (caractere, numérico, lógico e etc.) a um tipo complexo como um array ou um objeto. Desta forma 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 a 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 no 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, segue abaixo as diferentes definições para o componentes Dialog e MsDialog:

Exemplo 01 – Include Rwmake.ch

```
#include “rwmake.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
```

Ambas as sintaxes produzirão o mesmo efeito quando compiladas e executadas no ambiente Protheus, mas deve ser utilizada sempre a sintaxe ADVPL através 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(), o qual é utilizado como base para os demais componentes da interface visual, pois um componente MSDIALOG() é uma janela da aplicação.

MSGET()

Sintaxe	**@ nLinha, nColuna MSGET VARIAVEL SIZE nLargura,nAltura UNIDADE OF oObjetoRef F3 cF3 VALID VALID WHEN WHEN PICTURE cPicture**
Descrição	Define o componente visual MSGET, o qual 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, o qual é utilizado para 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, 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.

Interface visual completa

Abaixo segue um código completo de interface, utilizado 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 utilizados 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 os modelos utilizam como premissa que a estrutura da tabela a ser utilizada esteja definida no dicionário de dados do sistema (SX3).

13.2.1. 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**

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 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, o qual deve estar definido no dicionário de dados – SX3.

☑ **Variáveis private adicionais**

aRotina	Array contendo as funções que serão executadas pela Mbrowse. Este array pode ser parametrizados 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.

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"    ,"AxDeleta",0,5})

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

Importante	1. Utilizando a parametrização exemplificada, o efeito obtido com o uso da Mbrowse() será o mesmo obtido com o uso da AxCadastro(). 2. 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; 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 BInclui() 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"    ,"AxDeleta"    ,0,5})

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

USER FUNCTION BInclui(cAlias, nReg, nOpc)

Local cTudoOk := "(Alert('OK'),.T.)"
AxInclui(cAlias,nReg,nOpc,,,,cTudoOk)

RETURN
```

Anotações

13.2.3. 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()**

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, 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	**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().

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:

```
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”:

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
```

Importante

Para 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_CRCONV 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 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

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			

- ☑ 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;
- ☑ 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 ou para clientes devem ter sua identificação iniciada por um caractere "_".

LISTAS DE EXERCÍCIOS

Módulo 01: Introdução à programação

Exercícios

01	Aprimorar os seguintes algoritmos descritos na apostila: ☑ Usar telefone público – cartão ☑ Fritar um ovo ☑ Mascar um chiclete ☑ Trocar lâmpadas ☑ Descascar batatas ☑ Jogar o "Jogo da Forca"

Anotações

02	Montar os diagramas de blocos para os algoritmos desenvolvidos no exercício anterior: ☑ Usar telefone público – cartão ☑ Fritar um ovo ☑ Mascar um chiclete ☑ Trocar lâmpadas ☑ Descascar batatas ☑ Jogar o "Jogo da Forca"

Anotações

Exercícios

01	Desenvolver um programa que implemente o algoritmo de descascar batatas, utilizando a estrutura de repetição While, demonstrando quantas batatas foram descascadas:

Anotações

02	Desenvolver um programa que implemente o algoritmo de descascar batatas, utilizando a estrutura de repetição FOR, demonstrando quantas batatas foram descascadas:

Anotações

03	Desenvolver um programa que implemente o algoritmo de descascar batatas, utilizando a estrutura de repetição FOR, demonstrando quantas batatas faltam para serem descascadas:

Anotações

04	Desenvolver um programa que implemente o algoritmo do Jogo da Forca:

Anotações

05	Desenvolver um programa que implemente o algoritmo do Jogo da Velha:

Anotações

06	Desenvolver um programa que permita ao usuário pesquisar um cliente informando seu CNPJ e caso o mesmo exista na base, exibir suas principais informações.

Anotações

07	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.

Anotações

08	Utilizando a função AVISO() desenvolver um programa que permita ao usuário selecionar a opção de busca de CNPJ por cliente ou fornecedor, e caso encontre exiba seus dados principais.

Anotações

09	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.

Anotações

10	Utilizando a função FORMBATCH() desenvolver uma rotina que verifique se para cada item de um nota fiscal de entrada existe o respectivo cabeçalho, e caso seja encontrado algum item inconsistente, comunique esta ocorrência ao usuário que está realizando o processamento.

Anotações

11	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.

Anotações

Exercícios

01	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 nesta tabela relacionada.

Anotações

02	Desenvolver uma validação para um campo caractere existente na base, para que seja avaliado se aquele código já existe cadastrado, e caso positivo exiba uma mensagem de aviso alertando desta ocorrência.

Anotações

03	Desenvolver um gatilho que retorne uma descrição complementar para um campo vinculado ao campo código utilizado nos exercícios anteriores.

Anotações

04	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.

Anotações _____

05	Proteger a rotina desenvolvida no exercício anterior, de forma a garantir que na utilização da filial como prefixo do título não irá ocorrer duplicidade de dados no contas a pagar do financeiro.

Anotações _____

06	Implementar uma validação adicional no cadastro de clientes, através do ponto de entrada adequado, de forma que o campo CNPJ (A1_CGC) seja obrigatório para todos os tipos de cliente, exceto os definidos como Exterior.

Anotações _____

MÓDULO 04: Desenvolvendo aplicações em ADVPL

18. 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 realizem 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 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. 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.

19. 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, 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 as 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 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 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.

19.1. 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. 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 onde 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
```

20. Declaração e Atribuição de Variáveis

20.1. 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 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 a **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 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.

20.2. 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 necessário mais se referenciar ao seu conteúdo, e sim ao seu nome.

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.

20.3. 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 deve-se 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 nNumero := 10

Esta linha de código declara uma variável chamada nNumero indicando que 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 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 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 mantêm seu valor atual quando a execução da função Pai termina. Entretanto, somente pode ser acessada por execuções subseqüentes 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, 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.

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 caracter "_" 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 *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.)
```

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. 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).

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 caracter "_" antes do nome de variáveis PRIVATE e PUBLIC. Maiores informações avaliar o tópico: Boas Práticas de Programação.

Exemplo: Public _cRotina

20.4. 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 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.

21. Regras adicionais da linguagem ADVPL

21.1. 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			

- ☑ 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;
- ☑ 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 ou para clientes devem ter sua identificação iniciada por um caractere "_".

21.2. Pictures de formatação disponíveis

Com base na documentação disponível no DEM – Documentação Eletrônica Microsiga, 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 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.

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: @!
```

22. 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).

22.1. 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:**

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: 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
```

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

RestArea(aArea)
Return lRet
```

22.2. 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.

22.2.1. 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 declarado 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,* [*lMensApag*] , [*aBotoes*])**
- ☑ **Parâmetros:**

oDlg	Objeto	Janela onde a barra será criada.
bOk	Objeto	Bloco de código executado quando clicado botão Ok.
bCancelar	Objeto	Bloco de código executado quando clicado.
lMensApag	Lógico	Indica se ao clicar no botão Ok aparecerá uma tela de confirmação de exclusão. Valor padrão falso.
aBotões	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], [lPar6], [anClrText], [anClrBack], [anWidth], [anHeight], [alLowered], [alRaised])**

- ☑ **Parâmetros:**

nRow	Numérico vertical em pixel.
nCol	Numérico horizontal em pixel.
cText	Texto a ser exibido ao fundo.
oWnd	Objeto da janela ou controle onde será criado o objeto.
oFont	Características da fonte do texto que aparecerá ao fundo.
lCentered	Exibe o texto do título centralizado.
lPar6	Reservado.
nClrText	Cor do texto de controle.
nClrBack	Cor do fundo de controle.
nWidth	Largura do controle em pixel.
nHeight	Altura do controle em pixel.
lLowered	Exibe o painel rebaixado em relação ao controle de fundo.
lRaised	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], [lPar7], [lPar8], [lPar9], [alPixels], [anClrText], [anClrBack], [anWidth], [anHeight], [lPar15], [lPar16], [lPar17], [lPar18], [lPar19])**

☑ **Parâmetros:**

anRow	Numérico, opcional. Coordenada vertical em pixels ou caracteres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
abText	Code-Block, opcional. Quando executado deve retornar uma cadeia de caracteres a ser exibida.
aoWnd	Objeto, opcional. Janela ou diálogo onde o controle será criado.
acPicture	Caractere, opcional. Picture de formatação do conteúdo a ser exibido.
aoFont	Objeto, opcional. Objeto tipo tFont para configuração do tipo de fonte que será utilizado para exibir o conteúdo.
lPar7	Reservado.
lPar8	Reservado.
lPar9	Reservado.
alPixels	Lógico, opcional. Se .T. considera coordenadas passadas em pixels se .F., padrão, considera as coordenadas passadas em caracteres.
anClrText	Numérico, opcional. Cor do conteúdo do controle.
anClrBack	Numérico, opcional. Cor do fundo do controle.
anWidth	Numérico, opcional. Largura do controle em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
lPar15	Reservado.
lPar16	Reservado.
lPar17	Reservado.
lPar18	Reservado.
lPar19	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], [lPar12], [oPar13], [alPixel], [cPar15], [lPar16], [abWhen], [lPar18], [lPar19], [abChange], [alReadOnly], [alPassword], [cPar23], [acReadVar], [cPar25], [lPar26], [nPar27], [lPar28])**

☑ **Parâmetros:**

anRow	Numérico, opcional. Coordenada vertical em pixels ou caracteres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
abSetGet	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.
aoWnd	Objeto, opcional. Janela ou controle onde o controle será criado.
anWidth	Numérico, opcional. Largura do controle em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
acPict	Caractere, opcional. Máscara de formatação do conteúdo a ser exibido.
abValid	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.
anClrFore	Numérico, opcional. Cor de fundo do controle.
anClrBack	Numérico, opcional. Cor do texto do controle.
aoFont	Objeto, opcional. Objeto tipo tFont utilizado para definir as características da fonte utilizada para exibir o conteúdo do controle.
lPar12	Reservado.
oPar13	Reservado.
alPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
cPar15	Reservado.
lPar16	Reservado.
abWhen	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.
lPar18	Reservado.

lPar19	Reservado.
abChange	Bloco de código, opcional. Executado quando o controle modifica o valor da variável associada.
alReadOnly	Lógico, opcional. Se .T. o controle não poderá ser editado.
alPassword	Lógico, opcional. Se .T. o controle exibirá asteriscos "*" no lugar dos caracteres exibidos pelo controle para simular entrada de senha.
cPar23	Reservado.
acReadVar	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.
lPar26	Reservado.
nPar27	Reservado.
lPar18	Reservado.

Objeto MsGetDados()

Objeto tipo lista com uma ou mais colunas para cadastramento de dados baseado em um vetor. Sua utilização exige que seja utilizado 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:**

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

Elemento	Conteúdo
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 está 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.).

Anotações

22.2.2. 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	**Função principal;**
2	Declaração e atribuição de variáveis;
3	Acesso a 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	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 referente a 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 e 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	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	**Fim da função de inclusão.**

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

      ACTIVATE MSDIALOG oDlg CENTER ON INIT ;
      EnchoiceBar(oDlg,{|| IIF(U_Mod2TOk(), Mod2GrvI(),;
      ( 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 | Mod2GrvI | Autor | Robson Luiz (rleg) | Data | 01.01.2007 |
//+-------------------------------------------------------------------+
//| Descr. | Rotina para gravar os dados na inclusão.                 |
//+-------------------------------------------------------------------+
//| Uso    | Para treinamento e capacitação.                          |
//+-------------------------------------------------------------------+
Static Function Mod2GrvI()
      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 nI := 0

      dbSelectArea("ZA3")
      For nI := 1 To Len( aCOLS )
            dbGoTo(aREG[nI])
            RecLock("ZA3",.F.)
            dbDelete()
            MsUnLock()
      Next nI
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")])
                  MsgAlert(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 )
```

22.2.3. Função Modelo2()

A função Modelo2() é uma interface pré-definida pela Microsiga que implementa de forma padronizada os compontes necessários a 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 Modelo2() estar vinculada ao uso de uma MBrowse().

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

☑ **Parâmetros:**

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

☑ **Retorno:**

Lógico	Indica se a tela da interface Modelo2() foi confirmada ou cancelada pelo usuário.

Anotações

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." // Validacao para a exclusao. 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   := {}
```

Exemplo (continuação):

```
// 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
```

22.3. 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 desmistifica 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 apresentado aqui toda a explicação e 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:**

cAlias	Alias do dados a serem cadastrados.
nReg	Número do registro da tabela a ser editado.
uPar1	Parâmetro reservado.
uPar2	Parâmetro reservado.
uPar3	Parâmetro reservado.
aAcho	Vetor com os campos que serão apresentados pela MsMGet.
aPos	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.
aCpos	Vetor com os campos que poderão ser alterados.
uPar4	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>+...".
uPar5	Parâmetro reservado.
uPar6	Parâmetro reservado.
uPar7	Parâmetro reservado.
oWnd	Objeto no qual a MsMGet será criada.
uPar8	Parâmetro reservado.
lMemoria	Indica se será usado variáveis de memória ou os campos da tabela para cadastramento dos dados. Valor padrão falso.
lColuna	Indica se a MsMGet será apresentada com um objeto por linha (uma coluna). Valor padrão falso. Parâmetro reservado.
uPar9	Parâmetro reservado.
lSemPastas	Indica se não será usado as Pastas de Cadastro na MsMGet. Função executada para validar a exclusão de uma linha do aCols.

Vale lembrar que nós programadores reaproveitamos muito o que já existe, isto é para simplesmente ganharmos 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 dela. Então fica assim o procedimento:

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

E em outro momento aproveitamos está 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.

Anotações

22.3.1. 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 a 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 e 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.**

Anotações

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
```

```
//+-------------------------------------------------------------------+
//| 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 := MSGetDados():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
```

```
//+-------------------------------------------------------------------+
//| 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 := MSGetDados():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 nI := 0

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

      dbSelectArea( cAlias )
      dbSetOrder(1)
      dbSeek( xFilial( cAlias ) + cChave )
```

Continuação:

```
      While !EOF() .And. ZA2->( ZA2_FILIAL + ZA2_NUM ) == xFilial( cAlias ) +
cChave
            AADD( aREG, ZA2->( 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("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 nI := 0
nTotal := 0
For nI := 1 To Len( aCOLS )
      If aCOLS[nI,Len(aHeader)+1]
           Loop
      Endif
      nTotal+=Round(aCOLS[nI,GdFieldPos("ZA2_QTDVEN")]*;
      aCOLS[nI,GdFieldPos("ZA2_PRCVEN")],2)
Next nI
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 nI := 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. lRet
      MsgAlert("Campo PRODUTO preenchimento obrigatorio",cCadastro)
      lRet := .F.
   Endif
   If Empty(aCOLS[nI,GdFieldPos("ZA2_QTDVEN")]) .And. lRet
      MsgAlert("Campo QUANTIDADE preenchimento obrigatorio",cCadastro)
      lRet := .F.
   Endif
   If Empty(aCOLS[nI,GdFieldPos("ZA2_PRCVEN")]) .And. lRet
      MsgAlert("Campo PRECO UNITARIO preenchimento obrigatorio",cCadastro)
      lRet := .F.
   Endif

   If !lRet
      Exit
   Endif
Next i
Return( lRet )
```

Função para efetuar a gravação dos dados em ZA1 e ZA2 na inclusão, alteração e exclusão.

```
//+-----------------------------------------------------------------------+
//| 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 nI := 1 To Len( aHeader )
                        FieldPut( FieldPos( Trim( aHeader[nI, 2] ) ),
aCOLS[nX,nI] )
                  Next nI
                  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
```

Continuação:

```
                  If !aCOLS[ nX, Len( aHeader ) + 1 ]
                        RecLock( "ZA2", .T. )
                  Endif
            Endif

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

22.3.2. Função Modelo3()

A função Modelo3) é uma interface pré-definida pela Microsiga que implementa de forma padronizada os compontes necessários 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()**

- 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:**

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

☑ **Retorno:**

Lógico	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." // Validacao para a exclusao. 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  := "AllwaysTrue()"
Local cTudOk  := "AllwaysTrue()"
Local cFieldOk:= "AllwaysTrue()"
Local aCposE  := {}
Local nUsado, nX  := 0
```

Exemplo (continuação):

```
//ÚÄÄÄ
```

Exemplo (continuação):

```
If Len(aCols)>0
      //ÚÄÄÄ
```

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

☑ **Parâmetros:**

nLin1	Número da Linha Inicial
nCol1	Número da Coluna Inicial
nLin2	Número da Linha Final
nCol2	Número da Coluna Final
cAlias	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.
aFixe	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. Arquivos que fazem parte do dicionários de dados [n][1]=>Descrição do campo [n][2]=>Nome do campo Arquivos de trabalho [n][1]=>Descrição do campo [n][2]=>Nome do campo [n][3]=>Tipo [n][4]=>Tamanho [n][5]=>Decimal [n][6]=>Picture

☑ **Parâmetros:**

cCpo	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 **cCpo**, **cFun** e **aColors**.
nPar08	Parâmetro reservado.
cFun	Função que retornará um valor lógico para exibição do bitmap de status. Quando esse parâmetro é utilizado, o parâmetro **cCpo** é automaticamente desconsiderado.
nClickDef	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.
aColors	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 resource do repositório e quando esse parâmetro é utilizado os parâmetros **cCpo** e **cFun** são automaticamente desconsiderados.
cTopFun	Função que retorna o limite superior do filtro baseado na chave de índice selecionada. Esse parâmetro deve ser utilizado em conjunto com o parâmetro **cBotFun**.
cBotFun	Função que retorna o limite inferior do filtro baseado na chave de índice selecionada. Esse parâmetro deve ser utilizado em conjunto com o parâmetro **cTopFun**.
nPar14	Parâmetro reservado.
bInitBloc	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.
lNoMnuFilter	Valor lógico que define se a opção de filtro será exibida no menu da MBrowse. .T. => Não exibe a opção no menu .F. => (default) Exibe a opção no menu. A opção de filtro na MBrowse está disponível apenas para TopConnect.
lSeeAll	Identifica se o Browse deverá mostrar todas as filiais. O valor default é .F. (False), não mostra todas as filiais. Caso os parâmetros **cTopFun** ou **cBotFun** sejam informados esse parâmetro será configurado automaticamente para .F. (False) Parâmetro válido à partir da **versão 8.11**. A função **SetBrwSeeAll** muda o valor default desse parâmetro.
lChgAll	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 **lSeeAll** é configurado automaticamente para .T. (True). Caso os parâmetros **cTopFun** ou **cBotFun** sejam informados esse parâmetro será configurado automaticamente para .F. (False). Parâmetro válido à partir da **versão 8.11**. A função **SetBrwChgAll** muda o valor default desse parâmetro.

☑ **Variáveis private adicionais**

aRotina	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:

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" ,"AxDeleta",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" ,"AxDeleta" ,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 nOpcao       := 0

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

If nOpcao == 1
      MsgInfo(“Inclusão concluída com sucesso!”)
ElseIf       == 2
      MsgInfo(“Inclusão cancelada!”)
Endif

Return Nil
```

Anotações

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 especifica chamando a AxDeleta              |
+-----------------------------------------------------------------------------
| Uso       | Curso ADVPL                                                     |
+-----------------------------------------------------------------------------
/*/

User Function Exclui(cAlias, nReg, nOpc)

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

nOpcao := AxDeleta(cAlias,nReg,aRotina[nOpc,4])
// Identifica corretamente a opção definida para o função em aRotinas com mais
// do que os 5 elementos padrões.

If nOpcao == 1
      MsgInfo("Exclusão realizada com sucesso!")
ElseIf       == 2
      MsgInfo("Exclusão cancelada!")
Endif

Return Nil
```

22.4.1. 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	**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().

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().

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().

Anotações

22.4.2. FilBrowse()

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

☑ **Sintaxe: FilBrowse(cAlias, aQuery, cFiltro, lShowProc)**

☑ **Parâmetros:**

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

22.4.3. EndFilBrw()

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

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

☑ **Parâmetros:**

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

22.4.4. 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) }

22.4.5. 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

- 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 cFiltra := "A2_FILIAL == '"+xFilial('SA2')+"' .And. A2_EST == 'SP'"

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

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_BDeleta"      ,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)
```

Exemplo (continuação):

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

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
```

Exemplo (continuação):

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

Anotações

22.5. MarkBrowse()

A função MarkBrow() permite que os elementos de um browse, sejam marcados ou desmarcados. Para utilização da MarkBrow() 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, cExpIni, cExpFim, cAval, bParBloco)**

☑ **Parâmetros:**

cAlias	Alias ativo definido para a Mbrowse()
cCampo	Campo do arquivo onde será feito o controle (gravação) da marca.
cCpo	Campo onde será feita a validação para marcação e exibição do bitmap de status.
aCampos	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).
lInvert	Inverte a marcação.
cMarca	String a ser gravada no campo especificado para marcação.
cCtrlM	Função a ser executada caso deseje marcar todos elementos.
uPar8	Parâmetro reservado.
cExpIni	Função que retorna o conteúdo inicial do filtro baseada na chave de índice selecionada.
cExpFim	Função que retorna o conteúdo final do filtro baseada na chave de índice selecionada.
cAval	Função a ser executada no duplo clique em um elemento no browse.
bParBloco	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:

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.
cMarca	Marca em uso pela MarkBrw()
lInverte	Indica se foi utilizada a inversão da seleção dos itens no browse.

Exemplo: Função MarkBrow() e acessórias

```
#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"       ,"AxDeleta" ,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
      //            "123456789012345678901234567890123456789 0
      //            "CCCCCC |  LL  | NNNNNNNNNNNNNNNNNNNN +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 Selecionados" 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
      LimpaMarca()
ENDIF

RETURN
```

Exemplo: Função LimpaMarca() – utilização das funções acessórias da MarkBrow()

```
/*/
+-----------------------------------------------------------------------------
| 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
```

Anotações

23. Arquivos e Índices Temporários

23.1. 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 existe 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.

23.2. Funções para manipulação de arquivos e índices temporários

23.2.1. CriaTrab()

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

☑ **Sintaxe: CriaTrab(aCampo, lCriar, cExt)**

☑ **Parâmetros**

aCampo	Array com o nome, tipo, tamanho e decimal do campo a ser criado no arquivo
lCriar	Se verdadeiro (.T.) criar o arquivo, ou falso (.F.) somente retorna um nome aleatório
cExt	Qual extensão deverá ser criado o arquivo de trabalho

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

23.2.2. 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**

lNewArea	Indica se e um novo alias no conjunto de alias aberto
cDriver	Drive (RddName()) do arquivo -> DBFCDX / TOPCONN / DBFNTX
cName	Nome físico da tabela que será usado
cAlias	Alias que será usado enquanto estive aberto
lShared	A tabela terá acesso exclusivo ou compartilhado
lReadOnly	Se verdadeiro a tabela será somente leitura

23.2.3. 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, lShow)**

☑ **Parâmetros**

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

Anotações

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 nI          := 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 nI := 1 To Len( aStru )
      dbSeek( aStru[nI,2] )
      AADD( aArqTRB, { aStru[nI,1], X3_TIPO, X3_TAMANHO, X3_DECIMAL } )
Next nI
```

Exemplo (continuação):

```
// 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
```

- 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 no processamentos da rotina.

- 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
```

24. 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.

24.1.1. 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 Topconnect, porém não contemplando a tecnologia Protheus Embedded SQL.

Estrutura do programa

Linhas	Programa
1	**Função principal;**
2	Declaração e atribuição de variáveis;
3	Atualização do arquivo de perguntas através da função específica CriaSX1();
4	Definição as 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	**Fim da função principal.**
10	**Função de processamento e impressão do relatório**
11	Declaração e atribuição de variáveis;
12	Definição dos filtros de impressão, avaliando o bando 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	**Fim da função de processamento e impressão do relatório**
23	**Função de atualização do arquivo de perguntas**
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 PUTSX1() 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 ???? e o nome do usuário."

Private cString  := "SE2"
Private Tamanho  := "M"
Private aReturn  := { "Zebrado",2,"Administração",2,2,1,"",1 }
Private wnrel    := "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
*/
```

Continuação:

```
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,lCompres,;
//cSize,aFilter,lFiltro,lCrystal,cNameDrv,lNoAsk,lServer,cPortToPrint)

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

Continuação:

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

Continuação:

```
      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_PREFIXO, 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 //Titulo
      cOrder := "E2_NUM,E2_FORNECE,E2_LOJA"
Elseif aReturn[8] == 3 //Emissao
      cOrder := "E2_EMISSAO,E2_FORNECE,E2_LOJA"
Elseif aReturn[8] == 4 //Vencimento
      cOrder := "E2_VENCTO,E2_FORNECE,E2_LOJA"
```

Continuação:

```
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. !lEnd

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

Continuação:

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

While !Eof() .And. !lEnd .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
```

Continuação:

```
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")
      MsgAlert("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
***/
```

Continuação:

```
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],;
{},;
{},;
"")
Next i

Return
```

25. Manipulação de arquivos I

25.1. 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(), FClose(), 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().

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 utiliza a segunda família, representada pelas funções: FT_FUse(), FT_FGoTo(), FT_FRecno(), FT_FGoTop(), FT_FLastRec(), FT_FEof(), FT_FReadLn() e FT_FSkip().

1ª. Família de funções de gravação e leitura de arquivos texto

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
    MsgAlert("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)//AAAAMMDD
      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    | OKLETXT   | 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
    MsgAlert("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
```

Exemplo (continuação):

```
/*/
+-----------------------------------------------------------------------------
| 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    := ""

//123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
//000000000100000000020000000003000000000400000000050000000006000000000070
//FFCCCCCCLLNNNNNNNNNNNNNNNNNNNNNNNNVVVVVVVVVVVVVVVVVVVVDDDDDDDD
//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(nHdl,0,2)
fSeek(nHdl,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
```

2ª. Família de funções de gravação e leitura de arquivos 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 e 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
```

Exemplo (continuação):

```
//+-------------------------------------------
//| 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)
   MsgAlert("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() //LENDO 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)

   MsgInfo(cMsg)

   FT_FSKIP()   //próximo registro no arquivo txt
EndDo

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

26. Oficina de programação I

26.1. Interfaces com sintaxe clássica

A sintaxe convencional para definição de componentes visuais da linguagem ADVPL depende diretamente no 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, segue abaixo as diferentes definições para o componentes Dialog e MsDialog:

```
Exemplo 01 – Include Rwmake.ch

#include “rwmake.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
```

Ambas as sintaxes produzirão o mesmo efeito quando compiladas e executadas no ambiente Protheus, mas deve ser utilizada sempre 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()

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.

CHECKBOX()

Sintaxe	**@ nLinha,nColuna CHECKBOX oCheckBox VAR VARIAVEL PROMPT cTexto WHEN WHEN UNIDADE OF oObjetoRef SIZE nLargura,nAltura MESSAGE cMensagem**
Descrição	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()

Sintaxe	**@ nLinha,nColuna COMBOBOX VARIAVEL ITEMS AITENS SIZE nLargura,nAltura UNIDADE OF oObjetoRef**
Descrição	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()

Sintaxe	**@ nLinha,nColuna FOLDER oFolder OF oObjetoRef PROMPT &cTexto1,...,&cTextoX PIXEL SIZE nLargura,nAltura**
Descrição	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.

MSDIALOG()

Sintaxe	**DEFINE MSDIALOG oObjetoDLG TITLE cTitulo FROM nLinIni,nColIni TO nLiFim,nColFim OF oObjetoRef UNIDADE**
Descrição	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()

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

RADIO()

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**
Descrição	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 Radio pode conter um ou mais elementos.

SAY()

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

SBUTTON()

Sintaxe	**DEFINE SBUTTON FROM nLinha, nColuna TYPE N ACTION AÇÃO STATUS OF oObjetoRef**
Descrição	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.

26.2. Réguas 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()**

26.2.1. 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 acessórias

```
/*/
+-----------------------------------------------------------------------------
| 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( {|lEnd| RunProc(@lEnd)}, "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(lEnd)
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 lEnd
      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:**

nMaxProc	Variável que indica o valor máximo de processamento (passos) que serão indicados pela régua.

☑ **Retorno:**

Nenhum	.

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:**

cMensagem	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:**

Nenhum	.

Exemplo:

```
...

dbSelectArea("SA1")
dbGoTop()
SetRegua(LastRec())
While !Eof()
   IncRegua("Avaliando cliente:"+SA1->A1_COD)
   If Li > 60
...
```

Anotações

26.2.2. Processa()

Régua de processamento simples, com apenas um indicador de progresso, utilizada no processamento de rotinas.

☑ **Sintaxe: Processa(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 PROCESSA() e acessórias

```
/*/
+-----------------------------------------------------------------------------
| 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( {|lEnd| RunProc(@lEnd)}, "Aguarde...","Executando rotina.", .T. )

Return Nil

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

Static Function RunProc(lEnd)
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 lEnd
      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:**

nMaxProc	Variável que indica o valor máximo de processamento (passos) que serão indicados pela régua.

☑ **Retorno:**

Nenhum	.

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:**

cMensagem	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:**

Nenhum	.

Exemplo:

```
...
dbSelectArea("SA1")
dbGoTop()
SetProc(LastRec())
While !Eof()
   IncProc("Avaliando cliente:"+SA1->A1_COD)
   If Li > 60
...
```

26.2.3. 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:**

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	

☑ **Métodos:**

Activate()	Inicia a execução do objeto MsNewProcess instanciado.
SetRegua1()	Define a quantidade de informações que serão demonstradas pelo indicador de progresso superior. **Parâmetro**: nMaxProc
IncRegua1()	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(). **Parâmetro:** cMensagem
SetRegua2()	Define a quantidade de informações que serão demonstradas pelo indicador de progresso inferior. **Parâmetro:** nMaxProc
IncRegua2()	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(). **Parâmetro:** 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({|lEnd| RunProc(lEnd,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
```

```
Exemplo (continuação):

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 lEnd
         Exit
      Endif
      dbSkip()
   End
Next i
Return
```

26.2.4. MsAguarde().

Indicador de processamento sem incremento.

☑ **Sintaxe: Processa(bAcao, cMensagem, cTitulo)**

☑ **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.
cTitulo	Título da janela da régua de processamento.
Aparência	

Exemplo: MSAguarde()

```
/*/
+-----------------------------------------------------------------------------
| Função    | GMSAGUARDE | Autor | ROBSON LUIZ           | Data |            |
+-----------------------------------------------------------------------------
| Descrição | Programa que demonstra a utilização das funções MSAGUARDE()    |
|           | e MSPROCTXT()                                                  |
+-----------------------------------------------------------------------------
| Uso       | Curso ADVPL                                                    |
+-----------------------------------------------------------------------------
/*/

USER FUNCTION GMsAguarde()
PRIVATE lEnd := .F.

MsAguarde({|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
```

26.2.5. MsgRun().

Indicador de processamento sem incremento.

☑ **Sintaxe: Processa(cMensagem, cTitulo, bAcao)**

☑ **Retorno: Nil**

☑ **Parâmetros:**

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

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
```

26.3.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 aHeader (array de cabeçalho).

26.3.1. ListBox simples

☑ **Sintaxe:**

```
@ nLinha,nColuna LISTBOX oListbox VAR nLista ITEMS aLista SIZE nLargura,nAltura
OF oObjetoRef UNIDADE ON CHANGE CHANGE
```

☑ **Parâmetros:**

nLinha,nColuna	Posição do objeto ListBox em função da janela em que ele será definido.
oListBox	Objeto ListBox que será criado.
nLista	Variável numérica que contém o número do item selecionado no ListBox.
aLista	Vetor simples contendo as strings que serão exibidas no ListBox.
nLargura,nAltura	Dimensões do objeto ListBox.
oObjetoRef	Objeto dialog no qual o componente será definido.
UNIDADE	Unidade de medida das dimensões: PIXEL.
CHANGE	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 ListBoxIte()

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
```

26.3.2. ListBox múltiplas colunas

☑ **Sintaxe:**

@ nLinha,nColuna LISTBOX oListbox FIELDS HEADER "Header1", ..., "HeaderX" SIZE nLargura,nAltura OF oObjetoRef UNIDADE

☑ **Parâmetros:**

nLinha,nColuna	Posição do objeto ListBox em função da janela em que ele será definido.
oListBox	Objeto ListBox que será criado.
nLista	Variável numérica que contém o número do item selecionado no ListBox.
"Header1",...,"HeaderX"	Strings identificando os títulos das colunas do Grid.
nLargura,nAltura	Dimensões do objeto ListBox.
oObjetoRef	Objeto dialog no qual o componente será definido.
UNIDADE	Unidade de medida das dimensões: PIXEL.
CHANGE	Função ou lista de expressões que será executada na seleção de um item do ListBox.

☑ **Métodos:**

SetArray()	Método o objeto ListBox que define qual o array contém os dados que serão exibidos no grid.

☑ **Atributos:**

bLine	Atributo do objeto ListBox que vincula cada linha,coluna do array, com cada cabeçalho do grid.

☑ **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
```

26.4. ScrollBox()

O ScrollBox é o objeto utilizado para permitir a um Dialog exibir 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 Dialog, e os componentes ao ScrollBox.

- ❑ MsDialog() ← ScrollBox()
- ❑ ScrollBox() ← Componentes Visuais

☑ **Sintaxe:**

@ nLinha,nColuna SCROLLBOX oScrollBox HORIZONTAL VERTICAL SIZE nLargura,nAltura OF oObjetoRef BORDER

☑ **Parâmetros:**

nLinha,nColuna	Posição do objeto ScrollBox em função da janela em que ele será definido.
oScrollBox	Objeto ScrollBox que será criado.
HORIZONTAL	Parâmetro que quando definido habilita a régua de rolagem horizontal.
VERTICAL	Parâmetro que quando definido habilita a régua de rolagem vertical.
nLargura,nAltura	Dimensões do objeto ScrollBox.
oObjetoRef	Objeto dialog no qual o componente será definido.
BORDER	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
```

```
Exemplo (continuação):

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 Scrooll 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)
```

```
Exemplo (continuação):

      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.,.F.,.T.,,,;
GetTextWidth(0,Trim(aSay[n][1])),15,;
      .F.,.F.,.F.,.F.,.F.)
      oGet:=TGet():New(nTop-2,40,&cGet,oScroll,,7,cPict,,,,,.F.,,.T.,;
,.F.,,.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 Multa" ;
   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
```

Anotações

26.5. 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:**

aParamBox	Array de parâmetros de acordo com a regra da ParamBox
cTitulo	Titulo da janela de parâmetros
aRet	Array que será passado por referencia e retornado com o conteúdo de cada parâmetro
bOk	Bloco de código para validação do OK da tela de parâmetros
aButtons	Array contendo a regra para adição de novos botões (além do OK e Cancelar) // AADD(aButtons,{nType,bAction,cTexto})
lCentered	Se a tela será exibida centralizada, quando a mesma não estiver vinculada a outra janela
nPosx	Posição inicial -> linha (Linha final: nPosX+274)
nPosy	Posição inicial -> coluna (Coluna final: nPosY+445)
oMainDlg	Caso o ParamBox deva ser vinculado a uma outra tela
cLoad	Nome do arquivo aonde as respostas do usuário serão salvas / lidas
lCanSave	Se as respostas para as perguntas podem ser salvas
lUserSave	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 Radio
[4] : Array contendo as opções do Radio
[5] : Tamanho do Radio
[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 Radio
[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 Radio
[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 - Clausula 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

27. 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**

27.1. 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 subclasse 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 dramaticamente a repetição de código em um projeto ou programa, sendo uma das principais vantagens da abordagem de orientação a objetos.

27.2. 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 a Encapsulamento mais elegante e mais poderosa 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 reaproveitar 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.

27.3. O Modelo de Objetos (OMT)

Um modelo de objetos busca capturar a estrutura estática de um sistema mostrando os objetos existentes, seus relacionamentos, e 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.

27.3.1. 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 amplos. 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.

27.3.2. 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.

27.3.3. 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

27.3.4. 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.

28. 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.

28.1. Sintaxe e operadores para orientação a objetos

Palavras reservadas

- CLASS
- CONSTRUCTOR
- DATA
- ENDCLASS
- FROM
- METHOD
- SELF

CLASS

Descrição	Utilizada na declaração de uma classe de objetos, e para identificar a qual classe um determinado método está relacionado.
Sintaxe 1	CLASS <nome_da_classe>
Sintaxe 2	METHOD <nome_do_método> CLASS <nome_da_classe>

CONSTRUCTOR

Descrição	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.
Sintaxe	METHOD <nome_do_método()> CONSTRUCTOR

DATA

Descrição	Utilizada na declaração de um atributo da classe de objetos.
Sintaxe	DATA <nome_do_atributo>

ENDCLASS

Descrição	Utilizada na finalização da declaração da classe.
Sintaxe	ENDCLASS

FROM

Descrição	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.
Sintaxe	CLASS <nome_da_classe> FROM <nome_da_superclasse>

METHOD

Descrição	Utilizada na declaração do protótipo do método de uma classe de objetos, e na declaração do método efetivamente desenvolvido.
Sintaxe 1	METHOD <nome_do_método()>
Sintaxe 2	METHOD <nome_do_método(<parâmetros>)> CLASS <nome_da_classe>

SELF

Descrição	Utilizada principalmente pelo método construtor para retornar o objeto criado para a aplicação.
Sintaxe	Return SELF

Operadores específicos

:	Utilizado para referenciar um método ou um atributo de um objeto já instanciado.
Exemplo 1	cNome := oAluno**:sNome**
Exemplo 2	cNota := oAluno**:GetNota(cCurso)**

::	Utilizado pelos métodos de uma classe para referenciar os atributos disponíveis para o objeto.
Exemplo	METHOD GetNota(cCurso) CLASS ALUNO Local nPosCurso := 0 Local nNota := 0 nPosCurso := aScan(**::aCursos**,{|aCurso| aCurso[1] == cCurso}) IF nPosCurso > 0 nNota := **::aCursos**[nPosCurso][2] ENDIF Return nNota

28.2. 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 nIdade

METHOD Create() CONSTRUCTOR
METHOD SetNome()
METHOD SetIdade()

ENDCLASS
```

28.3. 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 nIdade

METHOD Create() CONSTRUCTOR
METHOD SetNome()
METHOD SetIdade()

ENDCLASS

METHOD Create(cNome, nIdade) 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 SetIdade()
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 := CalcIdade(dNascimento)
Return SELF

STATIC FUNCTION CalcIdade(dNascimento)
Local nIdade
nIdade := dDataBase - dNascimento
RETURN nIdade
```

Herança entre classes

Seguindo o principio 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.

29. 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**
  - Tsrvobject
  - **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
      - Tpageview
      - Tpanel
      - Tradmenu
      - Tsbrowse
      - Tsay
      - Tscrollbox
      - Tsimpleeditor
      - Tslider

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

BRGETDDB()

Descrição	Classe de objetos visuais do tipo Grid.

MSCALEND()

Descrição	Classe de objetos visuais do tipo Calendário.

MSCALENDGRID()

Descrição	Classe de objetos visuais do tipo Grade de Períodos.

MSSELBR()

Descrição	Classe de objetos visuais do tipo controle - Grid

MSWORKTIME()

Descrição	Classe de objetos visuais do tipo controle - Barra de Período.

SBUTTON()

Descrição	Classe de objetos visuais do tipo botão, o qual pode possuir imagens padrões associadas ao seu tipo.

TBAR()

Descrição	Classe de objetos visuais do tipo Barra Superior.

TBITMAP()

Descrição	Classe de objetos visuais que permite a exibição de uma imagem.

TBROWSEBUTTON()

Descrição	Classe de objetos visuais do tipo botão no formato padrão utilizado em browses da aplicação.

TBTNBMP()

Descrição	Classe de objetos visuais do tipo botão, o qual permite que seja vinculada uma imagem ao controle.

TBTNBMP2()

Descrição	Classe de objetos visuais do tipo botão, o qual permite a exibição de uma imagem ou de um popup.

TBUTTON()

Descrição	Classe de objetos visuais do tipo botão, o qual permite a utilização de texto para sua identificação.

TCBROWSE()

Descrição	Classe de objetos visuais do tipo Grid.

TCHECKBOX()

Descrição	Classe de objetos visuais do tipo controle - CheckBox.

TCOLORTRIANGLE()

Descrição	Classe de objetos visuais do tipo Paleta de Cores.

TCOMBOBOX()

Descrição	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()

Descrição	Classe de objetos visuais do tipo controle - Folder.

TGET()

Descrição	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()

Descrição	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()

Descrição	Classe de objetos visuais do tipo botão com hiperlink.

TIBROWSER()

Descrição	Classe de objetos visuais do tipo Página de Internet, sendo necessário incluir a clausula BrowserEnabled=1 no Config do Remote.INI

TLISTBOX()

Descrição	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()

Descrição	Classe de objetos visuais do tipo controle - Menu.

TMENUBAR()

Descrição	Classe de objetos visuais do tipo controle - Barra de Menu.

TMETER()

Descrição	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.

TMSGRAPHIC()

Descrição	Classe de objetos visuais do tipo controle - Gráfico.

TMSGBAR()

Descrição	Classe de objetos visuais do tipo controle - Rodapé.

TMULTIBTN()

Descrição	Classe de objetos visuais do tipo controle - Múltiplos botões.

TMULTIGET()

Descrição	Classe de objetos visuais do tipo controle - edição de texto de múltiplas linhas.

TOLECONTAINER()

Descrição	Classe de objetos visuais do tipo controle, a qual permite a criação de um botão vinculado a um objeto OLE.

TPAGEVIEW()

Descrição	Classe de objetos visuais do tipo controle, que permite a visualização de arquivos no formato gerado pelo spool de impressão do Protheus.

TPANEL()

Descrição	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()

Descrição	Classe de objetos visuais do tipo controle – TRadMenu, a qual permite criar um controle visual no formato Radio Button.

TSBROWSE()

Descrição	Classe de objetos visuais do tipo controle – TSBrowse, a qual permite criar um controle visual do tipo Grid.

TSAY()

Descrição	Classe de objetos visuais do tipo controle – tSay, a qual exibe o conteúdo de texto estático sobre uma janela ou controle previamente definidos.

TSCROLLBOX()

Descrição	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()

Descrição	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()

Descrição	Classe de objetos visuais do tipo controle – tSlider, a qual permite criar um controle visual do tipo botão deslizante.

TSPLITTER()

Descrição	Classe de objetos visuais do tipo controle – tSplitter, a qual permite criar um controle visual do tipo divisor.

TTABS()

Descrição	Classe de objetos visuais do tipo controle – TTabs, a qual permite criar um controle visual do tipo pasta.

TTOOLBOX()

Descrição	Classe de objetos visuais do tipo controle – tToolbox, a qual permite criar um controle visual para agrupar diferentes objetos.

TWBROWSE()

Descrição	Classe de objetos visuais do tipo controle – TWBrowse, a qual permite criar um controle visual do tipo Grid.

VCBROWSE()

Descrição	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.

Para visualizar a documentação completa de todos os componentes mencionados neste capítulo deve ser acesso o site DEM – Documentação Eletrônica Microsiga (dem.microsiga.com.br) conforme abaixo:

29.1. Particularidades dos componentes visuais

29.1.1. 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()

nClrPane	Cor de fundo do painel
nClrText	Cor da fonte das letras do painel

TSAY()

nClrPane	Cor de fundo do painel
nClrText	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)

nRed	Valor de 0-255 para o elemento vermelho da paleta RGB
nGreen	Valor de 0-255 para o elemento verde da paleta RGB
nBlue	Valor de 0-255 para o elemento azul da paleta RGB
Retorno	Valor a ser definido para o atributo cor do componente

30. 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**

30.1. 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"):

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().

30.1.1. Enchoice()

☑ **Sintaxe: Enchoice(cAlias, nReg, nOpc, aCRA, cLetra, cTexto, aAcho, aPos, aCpos, nModelo, nColMens, cMensagem, cTudoOk, oWnd, lF3, lMemoria, lColumn, caTela, lNoFolder, lProperty)**

☑ **Retorno: Nil**
☑ **Parâmetros:**

cAlias	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada
nReg	Parâmetro não utilizado
nOpc	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização)
aCRA	Parâmetro não utilizado
cLetra	Parâmetro não utilizado
cTexto	Parâmetro não utilizado
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"
aPos	Vetor com coordenadas para criação da enchoice no formato {<top>, <left>, <bottom>, <right>}
aCpos	Vetor com nome dos campos que poderão ser editados
nModelo	Se for diferente de 1 desabilita execução de gatilhos estrangeiros
nColMens	Parâmetro não utilizado
cMensagem	Parâmetro não utilizado
cTudoOk	Expressão para validação da Enchoice
oWnd	Objeto (janela, painel, etc.) onde a enchoice será criada.
lF3	Indica se a enchoice esta sendo criada em uma consulta F3 para utilizar variáveis de memória
lMemoria	Indica se a enchoice utilizará variáveis de memória ou os campos da tabela na edição
lColumn	Indica se a apresentação dos campos será em forma de coluna
caTela	Nome da variável tipo "private" que a enchoice utilizará no lugar da propriedade aTela
lNoFolder	Indica se a enchoice não irá utilizar as Pastas de Cadastro (SXA)
lProperty	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    := {}
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 aTELA[0][0]
Private aGETS[0]

DbSelectArea("SX3")
DbSetOrder(1)
DbSeek(cAliasE)
```

```
Exemplo (continuação):

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*/,/*cTudoOk*/, oDlg, lF3, lMemoria, lColumn,;
        caTela, lNoFolder, lProperty)

ACTIVATE MSDIALOG oDlg CENTERED

Return
```

30.1.2. MsMGet()

☑ **Sintaxe: MsMGet():New(cAlias, nReg, nOpc, aCRA, cLetra, cTexto, aAcho, aPos, aCpos, nModelo, nColMens, cMensagem, cTudoOk, oWnd, lF3, lMemoria, lColumn, caTela, lNoFolder, lProperty)**

☑ **Retorno: oMsMGet → objeto do tipo MsMGet()**

☑ **Parâmetros:**

cAlias	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada
nReg	Parâmetro não utilizado
nOpc	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização)
aCRA	Parâmetro não utilizado
cLetra	Parâmetro não utilizado
cTexto	Parâmetro não utilizado
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"
aPos	Vetor com coordenadas para criação da enchoice no formato {<top>, <left>, <bottom>, <right>}
aCpos	Vetor com nome dos campos que poderão ser editados
nModelo	Se for diferente de 1 desabilita execução de gatilhos estrangeiros
nColMens	Parâmetro não utilizado
cMensagem	Parâmetro não utilizado
cTudoOk	Expressão para validação da Enchoice
oWnd	Objeto (janela, painel, etc.) onde a enchoice será criada.
lF3	Indica se a enchoice esta sendo criada em uma consulta F3 para utilizar variáveis de memória

lMemoria	Indica se a enchoice utilizará variáveis de memória ou os campos da tabela na edição
lColumn	Indica se a apresentação dos campos será em forma de coluna
caTela	Nome da variável tipo "private" que a enchoice utilizará no lugar da propriedade aTela
lNoFolder	Indica se a enchoice não irá utilizar as Pastas de Cadastro (SXA)
lProperty	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 aTELA[0][0]
Private aGETS[0]
```

Exemplo (continuação):

```
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,lF3,lMemoria,lColumn, caTela,;
         lNoFolder, lProperty)

oDlg:lCentered := .T.
oDlg:Activate()

Return
```

30.2. 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.

30.2.1. 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:**

nTop	Distancia entre a MsGetDB e o extremidade superior do objeto que a contém.
nLeft	Distancia entre a MsGetDB e o extremidade esquerda do objeto que a contém.
nBottom	Distancia entre a MsGetDB e o extremidade inferior do objeto que a contém.
nRight	Distancia entre a MsGetDB e o extremidade direita do objeto que a contém.
nOpc	Posição do elemento do vetor aRotina que a MsGetDB usará como referência.
cLinhaOk	Função executada para validar o contexto da linha atual do aCols.
cTudoOk	Função executada para validar o contexto geral da MsGetDB (todo aCols).
cIniCpos	Nome dos campos do tipo caracter que utilizarão incremento automático. Este parâmetro deve ser no formato "+<nome do primeiro campo>+<nome do segundo campo>+...".
lDelete	Habilita a opção de excluir linhas do aCols. Valor padrão falso.
aAlter	Vetor com os campos que poderão ser alterados.
nFreeze	Indica qual coluna não ficara congelada na exibição.
lEmpty	Habilita validação da primeira coluna do aCols para esta não poder estar vazia. Valor padrão falso.
uPar1	Parâmetro reservado.
cFieldOk	Função executada na validação do campo.
cTRB	Alias da tabela temporária.
lCondicional	Reservado
lAppend	Indica se a MsGetDB ira criar uma linha em branco automaticamente quando for inclusão.
cDelOk	Função executada para validar a exclusão de uma linha do aCols.
lDisparos	Indica se será utilizado o Dicionário de Dados para consulta padrão, inicialização padrão e gatilhos.
uPar2	Parâmetro reservado.
cSuperDel	-Função executada quando pressionada as teclas <Ctrl>+<Delete>.
oWnd	Objeto no qual a MsGetDB será criada.

☑ **Aparência:**

☑ **Variáveis private:**

aRotina	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
aHeader	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.
lRefresh	Variável tipo lógica para uso reservado.

☑ **Variáveis públicas:**

nBrLin	Indica qual a linha posicionada do aCols.

☑ **Funções de validação:**

cLinhaOk	Função de validação na mudança das linhas da grid. Não pode ser definida como Static Function.
cTudoOk	Função de validação da confirmação da operação com o grid. Não pode ser definida como Static Function.

☑ **Métodos adicionais:**

ForceRefresh()	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 nI
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", "AxDeleta", 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:lCentered := .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.
```

30.2.2. 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:**

nTop	Distancia entre a MsGetDados e o extremidade superior do objeto que a contém.
nLeft	Distancia entre a MsGetDados e o extremidade esquerda do objeto que a contém.
nBottom	Distancia entre a MsGetDados e o extremidade inferior do objeto que a contém.
nRight	Distancia entre a MsGetDados e o extremidade direita do objeto que a contém.
nOpc	Posição do elemento do vetor aRotina que a MsGetDados usará como referencia.
cLinhaOk	Função executada para validar o contexto da linha atual do aCols.
cTudoOk	Função executada para validar o contexto geral da MsGetDados (todo aCols).
cIniCpos	Nome dos campos do tipo caracter que utilizarão incremento automático. Este parâmetro deve ser no formato "+<nome do primeiro campo>+<nome do segundo campo>+...".
lDelete	Habilita excluir linhas do aCols. Valor padrão falso.
aAlter	Vetor com os campos que poderão ser alterados.
uPar1	Parâmetro reservado.
lEmpty	Habilita validação da primeira coluna do aCols para esta não poder estar vazia. Valor padrão falso.
nMax	Número máximo de linhas permitidas. Valor padrão 99.
cFieldOk	Função executada na validação do campo.
cSuperDel	Função executada quando pressionada as teclas <Ctrl>+<Delete>.
uPar2	Parâmetro reservado.
cDelOk	Função executada para validar a exclusão de uma linha do aCols.
oWnd	Objeto no qual a MsGetDados será criada.

☑ **Aparência:**

☑ **Variáveis private:**

aRotina	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
aHeader	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.
lRefresh	Variável tipo lógica para uso reservado.

☑ **Variáveis públicas:**

N	Indica qual a linha posicionada do aCols.

☑ **Funções de validação:**

cLinhaOk	Função de validação na mudança das linhas da grid. Não pode ser definida como Static Function.
cTudoOk	Função de validação da confirmação da operação com o grid. Não pode ser definida como Static Function.

☑ **Métodos adicionais:**

ForceRefresh()	Atualiza a MsGetDados com a tabela e posiciona na primeira linha.
Hide()	Oculta a MsGetDados.
Show()	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 nI
Local oDlg
Local oGetDados
Local nUsado := 0
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", "AxDeleta", 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,;
                      "",;
                      "" })
EndIf
DbSkip()
End

AADD(aCols,Array(nUsado+1))

For nI := 1 To nUsado
    aCols[1][nI] := CriaVar(aHeader[nI][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.
```

30.2.3. 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, cIniCpos, aAlterGDa, nFreeze, nMax, cFieldOk, cSuperDel, cDelOk, oDLG, aHeader, aCols)**

☑ **Retorno: oMsGetDados → objeto do tipo MsNewGetDados()**

☑ **Parâmetros:**

nSuperior	Distancia entre a MsNewGetDados e o extremidade superior do objeto que a contem
nEsquerda	Distancia entre a MsNewGetDados e o extremidade esquerda do objeto que a contem
nInferior	Distancia entre a MsNewGetDados e o extremidade inferior do objeto que a contem
nDireita	Distancia entre a MsNewGetDados e o extremidade direita do objeto que a contem
nOpc	Operação em execução: 2- Visualizar, 3- Incluir, 4- Alterar, 5- Excluir
cLinOk	Função executada para validar o contexto da linha atual do aCols
cTudoOk	Função executada para validar o contexto geral da MsNewGetDados (todo aCols)
cIniCpos	Nome dos campos do tipo caracter que utilizarão incremento automático.
aAlterGDa	Campos alteráveis da GetDados
nFreeze	Campos estáticos na GetDados, partindo sempre da posição inicial da getdados aonde: 1- Primeiro campo congelado 2- Primeiro e segundo campos congelados...
nMax	Número máximo de linhas permitidas. Valor padrão 99
cFieldOk	Função executada na validação do campo
cSuperDel	Função executada quando pressionada as teclas <Ctrl>+<Delete>
cDelOk	Função executada para validar a exclusão de uma linha do aCols
oDLG	Objeto no qual a MsNewGetDados será criada
aHeader	Array a ser tratado internamente na MsNewGetDados como aHeader
aCols	Array a ser tratado internamente na MsNewGetDados como aCols

☑ **Aparência:**

☑ **Variáveis private:**

aRotina	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
aHeader	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.
lRefresh	Variável tipo lógica para uso reservado.

☑ **Variáveis públicas:**

N	Indica qual a linha posicionada do aCols.

☑ **Funções de validação:**

cLinhaOk	Função de validação na mudança das linhas da grid. Não pode ser definida como Static Function.
cTudoOk	Função de validação da confirmação da operação com o grid. Não pode ser definida como Static Function.

☑ **Métodos adicionais:**

ForceRefresh()	Atualiza a MsNewGetDados com a tabela e posiciona na primeira linha.
Hide()	Oculta a MsNewGetDados.
Show()	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  := {}
```

Exemplo (continuação):

```
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 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
```

```
Exemplo (continuação):

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,lF3, lMemoria,lColumn,caTela,lNoFolder,;
            lProperty)

      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:lCentered := .T.
oDlg:Activate()
Return
```

30.2.3.1. 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:**

lUseDefaultColors	Atributo que deverá ser definido como .F. para que as alterações nas cores sejam permitidas.

☑ **Métodos adicionais:**

SetBlkBackColor	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    | MRBWGTCL   | 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
```

```
Exemplo (continuação):

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
```

```
Exemplo (continuação):

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,lF3, lMemoria,lColumn,caTela,lNoFolder,lProperty)

      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({|| GETDCLR(oGetD:aCols,oGetD:nAt,aHeader)})

oDlg:bInit  := {|| EnchoiceBar(oDlg, {||oDlg:End()}, {||oDlg:End()},,aButtons)}
oDlg:lCentered := .T.
oDlg:Activate()

Return
```

```
Exemplo (continuação):

// 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
```

Anotações

30.3. 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.

30.3.1. EnchoiceBar()

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:**

```
EnchoiceBar( oDlg, bOk, bCancel, lMsgDel, aButtons, nRecno, cAlias)
```

☑ **Parâmetros:**

oDlg	Dialog onde irá criar a barra de botões
bOk	Bloco de código a ser executado no botão Ok
bCancel	Bloco de código a ser executado no botão Cancelar
lMsgDel	Exibe dialog para confirmar a exclusão
aButtons	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
nRecno	Registro a ser posicionado após a execução do botão Ok.
cAlias	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,{||lOk:=.T.,oDlg:End()},{||oDlg:End()},,@aButtons))

Return
```

Anotações

30.3.2. 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:**

oWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
nBtnWidth	Numérico, opcional. Largura do botão contido na barra
nBtnHeight	Numérico, opcional. Altura do botão contido na barra
l3D	Lógico, opcional. Define tipo da barra
cMode	Não utilizado.
oCursor	Objeto, opcional. Define Cursor ao posicionar o mouse sobre a barra.
cResource	Caracter, opcional. Imagem do recurso a ser inserido como fundo da barra.
lNoAutoAdjust	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:lCentered := .T.
oDlg:Activate()

Return
```

30.3.3. 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:**

DEFINE BUTTONBAR oBar SIZE nWidth, nHeight 3D MODE OF oDlg CURSOR

☑ **Retorno: ().**

☑ **Parâmetros:**

oBar	Objeto do tipo TBar() que será criado com a utilização da sintaxe ButtonBar().
nWidth	Numérico, opcional. Largura do botão contido na barra.
nHeight	Numérico, opcional. Altura do botão contido na barra.
3D	Se definido habilita a visualização em 3D da barra de botões.
oDlg	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
MODE	Define a forma de orientação do objeto ButtonBar utilizando os seguintes termos pré-definidos: TOP, BOTTOM, FLOAT
CURSOR	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:**

cBitMap	Nome da imagem disponível na aplicação.
oBar	Objeto do tipo TBar() no qual o botão será adicionado.
cAcao	Função ou lista de expressões que determina a ação que será realizada pelo botão.
cTexto	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
```

Exemplo (continuação):

```
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
```

Anotações

30.3.4. 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:**

cBitMap	Nome da imagem pré-definida existente na aplicação ERP que será vinculada ao botão.
bAcao	Bloco de código que define a ação que será executada com a utilização do botão.
cTexto	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:**

	DESTINOS		**DISCAGEM**
	EDIT		**EDITABLE**
	EXCLUIR		**FORM**
	GRAF2D		**GRAF3D**
	LINE		**NOTE**
	OBJETIVO		**OK**
	PENDENTE		**PRECO**
	PRODUTO		**S4SB014N**
	S4WB001N		**S4WB005N**
	S4WB006N		**S4WB007N**
	S4WB008N		**S4WB009N**
	S4WB010N		**S4WB011N**
	S4WB013N		**S4WB014A**
	S4WB016N		**SIMULACA**
	VENDEDOR		**USER**

☑ **Exemplo:**

```
AADD(aButtons,{"USER",{||AllwaysTrue()},"Usuário"})
```

APÊNDICES

BOAS PRÁTICAS DE PROGRAMAÇÃO

31. 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
```

32. 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”:

33. 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”:

- ☑ **dbSeek()**
- ☑ **dbSelectArea()**

33.1. Palavras em maiúsculo

A regra é utilizar caracteres em maiúsculo para:

☑ **Constantes:**

```
#define NUMLINES 60 #define NUMPAGES 1000
```

☑ **Variáveis de memória:**

```
M-> CT2_CRCONV M->CT2_MCONVER := CriaVar("CT2_CONVER")
```

☑ **Campos:**

```
SC6->C6_NUMPED
```

☑ **Querys:**

```
SELECT * FROM...
```

34. 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

35. 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(lDefault)
Local lOk
lOk := GetOk(lDefault)
If lOk
   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(lDefault)
Return If( GetOk(lDefault), .T., .F.)
```

O código acima ainda pode ser aprimorado conforme abaixo:

```
User Function GetAnswer(lDefault)
Return GetOk(lDefault)
```

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 10
<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 convêm 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 nNumero 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 referencia 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.

LISTAS DE EXERCÍCIOS

Módulo 04.1.: Desenvolvendo aplicações em ADVPL

Exercícios

01	Desenvolver um programa que implemente a função AxCadastro() para a tabela padrão do sistema SA2 – Cadastro de Fornecedores.

Anotações

02	Implementar no AxCadastro() desenvolvido no item anterior as funções de validação de alteração e validação de exclusão.

Anotações

03	Desenvolver um programa que implemente a função Mbrowse() utilizando as funções padrões de cadastro, para a tabela padrão do sistema SA2 – Cadastro de Fornecedores.

Anotações

04	Implementar no Mbrowse() desenvolvido no item anterior a função de legenda.

Anotações

05	Implementar no Mbrowse() desenvolvido no item anterior uma função específica de inclusão utilizando os recursos adicionais da função AxInclui().

Anotações

06	Desenvolver um programa que implemente uma MarkBrowse utilizando as funções padrões de cadastro, para a tabela padrão do sistema SA2 – Cadastro de Fornecedores.

Anotações

07	Implementar na MarkBrowse() desenvolvida no item anterior uma função para exclusão de múltiplos registros da tabela padrão do sistema SA2 – Cadastro de Fornecedores.

Anotações

08	Desenvolver um programa que implemente uma MarkBrowse() para a tabela padrão do sistema SA1 – Cadastro de Cliente, contemplando uma função que permita a visualização dos dados principais de múltiplos clientes selecionados.

Anotações

09	Desenvolver um programa que implemente uma Modelo2() para a tabela padrão do sistema SX5 – Cadastro de tabelas, contemplando a funcionalidade de inclusão.

Anotações

10	Implementar na Modelo2() desenvolvida no item anterior as funcionalidades de Visualização, Alteração e Exclusão.

Anotações

11	Desenvolver um programa que implemente uma Modelo3() para as tabelas padrões do sistema SF1 – Cabeçalho do Documento de Entrada e SD1 – Itens do documento de entrada, contemplando as funcionalidades de Visualização e Inclusão.

Anotações

12	Implementar na Modelo3() desenvolvida no item anterior as funcionalidades e Alteração e Exclusão.

Anotações

13	Desenvolver um programa que crie um arquivo de trabalho com estrutura similar ao SA1 e receba as informações desta tabela.

Anotações

14	Implementar na rotina desenvolvida no item anterior, um MarkBrowse que permite a seleção e visualização de apenas um registro selecionado.

Anotações

15	Desenvolver um programa que implemente um relatório que permita a impressão de uma listagem simples dos dados da tabela padrão SA1 – Cadastro de clientes.

Anotações

16	Desenvolver um programa que implemente um relatório que permita a impressão de uma listagem simples das notas fiscais e seus respectivos itens.

Anotações

17	Desenvolver um programa que realize a exportação dos itens marcados no SA1 – Cadastro de Clientes para um arquivo no formato .TXT em um diretório especificado pelo usuário.

Anotações

18	Desenvolver um programa que realize a importação das informações contidas em um arquivo no formato .TXT selecionado pelo usuário e as atualize no SA1 – Cadastro de Clientes.

Anotações

19	Desenvolver o mesmo programa do item anterior utilizando as funções da família FT.

Anotações

Exercícios

20	Desenvolver uma aplicação visual que implemente cada um dos componentes disponíveis no fonte DIALOG_OBJETOS.PRW.

Anotações

21	Implementar as rotinas de importação e exportação desenvolvidas nos itens anterior as réguas de processamento mais adequadas.

Anotações

22	Substituir em uma das rotinas desenvolvidas anteriormente a função Pergunte() pela função ParamBox().

Anotações

23	Desenvolver uma rotina que implementar um ListBox simples que permita visualizar as tabelas cadastradas no SX5 – Cadastro de Tabelas.

Anotações

25	Implementar na rotina desenvolvida no item anterior a opção de visualização dos itens de uma das tabelas selecionadas, demonstrando-os em uma ListBox Grid.

Anotações

26	Desenvolver uma rotina que implemente uma Modelo2() utilizando os componentes visuais necessários, complementados com os recursos de rolagem do objeto ScrollBox().

Anotações

Exercícios

27	Implementar a classe Aluno() com os métodos New(), BuscaAluno() e SalvaAluno().

Anotações

28	Desenvolver a interface visual para interagir com a classe Aluno() implementada anteriormente.

Anotações

29	Implementar na rotina desenvolvida no item anterior os recursos necessários para persistir as informações e recuperar os dados de itens previamente salvos.

Anotações

Exercícios

30	Desenvolver uma rotina que implemente uma Enchoice() para o SA1 – Cadastro de clientes.

Anotações

31	Adequar a rotina desenvolvida no item anterior para permitir o uso da classe de objetos MsMGet().

Anotações

32	Desenvolver uma interface de cadastro combinando os recursos de um objeto MsMGet() e um objeto MsNewGetDados() causando o mesmo efeito de uma interface no padrão da Modelo3().

Anotações

33	Implementar na rotina desenvolvida no item anterior o uso da função EnchoiceBar() com um botão que permita ao usuário visualizar um dos itens exibidos na interface.

Anotações

34	Adequar a rotina desenvolvida no item anterior de forma a substituir o uso da função EnchoiceBar() pela classe de objeto TBar().

Anotações

35	Converter a sintaxe clássica utilizada no fonte DIALOG_OBJETOS.PRW para a sintaxe orientada a objetos.

Anotações

36	Converter a sintaxe clássica dos fontes “ListBox” para a sintaxe orientada a objetos.

Anotações

Projeto: Avaliação prática do treinamento de ADVPL Intermediário

Objetivos do projeto

- ☑ Contemplar o conteúdo do curso e estimular a prática da programação utilizando a linguagem ADVPL;
- ☑ Acrescentar um grau de desafio ao treinamento;

Regras para entrega e apresentação

- ☑ Deverá ser entregue na data estabelecida pelo instrutor, contendo os fontes e demais arquivos necessários. A avaliação do projeto será realizada após a entrega, no prazo estabelecido pelo instrutor;
- ☑ Pode ser realizado em grupos de até 03 pessoas. Caso existam alunos sem grupos, estes poderão integrar equipes já formadas com o total de membros estabelecido, mediante aprovação do instrutor;

Avaliação do projeto

- ☑ Mesmo peso da prova teórica, podendo substituí-la integralmente.
- ☑ Serão considerados os quesitos:
  - ❑ Funcionalidades
  - ❑ Clareza da programação
  - ❑ Apresentação
  - ❑ Otimização de técnicas e recursos apresentados no treinamento

Funcionalidades que compõe o projeto

Desenvolver uma aplicação utilizando a linguagem ADVPL que contemple as seguintes funcionalidades:

01	Efetue a leitura de um arquivo em formato .TXT com os dados necessários para realizar o cadastro de fornecedores no sistema Protheus, e grave estas informações em um arquivo intermediário.
02	A partir da seleção dos registros disponíveis neste arquivo intermediário, o usuário poderá: ❑ Imprimir os itens selecionados; ❑ Efetivar os itens selecionados, gravando-os no SA2 – Cadastro de Fornecedores; ❑ Estornar itens já efetivados, eliminando-os do SA2 - Cadastro de Fornecedores; ❑ Excluir os registros selecionados do arquivo intermediário; ❑ Excluir todos os registros importados de um arquivo em formato .TXT especificado. ❑ Visualizar as informações do registro gerado no arquivo intermediário; ❑ Visualizar o registro gerado no SA2 – Cadastro de Fornecedores a partir de um item selecionado no arquivo intermediário.

MÓDULO 07: ADVPL Orientado à objetos II

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.

36. 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**
  - Tsrvobject
  - **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
      - Tpageview
      - Tpanel
      - Tradmenu
      - Tsbrowse
      - Tsay
      - Tscrollbox
      - Tsimpleeditor
      - Tslider

Atributos comuns as classes de componentes visuais

cCaption	Título ou conteúdo do objeto.
cF3	Consulta F3 ao qual o objeto está vinculado, quando o mesmo é do tipo get.
cMsg	Mensagem exibida na barra de status da janela principal quando o objeto ganha foco.
cName	Nome do objeto
cTooltip	Mensagem exibida quando objeto exibe seu tooltip.
cVariable	Variável que receberá o conteúdo selecionado ou digitado no objeto, quando o mesmo for to tipo get ou similar.
lPassword	Indica se o texto digitado no objeto será exibido no formato de password, quando o mesmo é do tipo get.
lReadOnly	Flag que indica se o objeto pode ou não ser editado.
lShowHint	Flag que ativa .T. ou desativa .F. a exibição do tooltip do objeto.
lVisibleControl	Se .T. o objeto é visível, se .F. o objeto é invisível.
nHeight	Altura em pixels.
nLeft	Coordenada horizontal em pixels.
nTop	Coordenada vertical em pixels.
nWidth	Largura em pixels.
Picture	Picture de digitação do objeto, quando o mesmo é do tipo get.

Eventos comuns as classes de componentes visuais

bAction	Bloco de código executado quando o objeto é utilizado, quando o mesmo é do tipo botão ou similar.
bChange	Bloco de código executado quando ocorre uma mundança na seleção de itens que compõe o objeto, como por exemplo em listas ou grids.
bF3	Bloco de código executado quando utilizado o recurso de consulta F3 vinculado ao objeto.
bGotFocus	Executado quando objeto ganha foco.
blClicked	Executado quando acionado click do botão esquerdo do mouse sobre o objeto.
blDblClick	Executado quando acionado duplo click do botão esquerdo do mouse sobre o objeto.
bLostFocus	Executado quando objeto perde foco.
brClicked	Executado quando acionado click do botão direito do mouse sobre o objeto.
bValid	Executado quando o conteúdo do objeto é modificado e deverá ser validado. Deve retornar .T. se o conteúdo é válido e .F. se conteúdo inválido.
bWhen	Executado quando há movimentação de foco na janela.Se retornar .T. o objeto continua habilitado, se retornar .F. o objeto será desabilitado.

TSRVOBJECT()

☑ **Descrição:** Classe abstrata inicial de todas as classes de interface do ADVPL. Não deve ser instanciada diretamente.

☑ **Propriedades:**

nLeft	Coordenada horizontal em pixels.
nTop	Coordenada vertical em pixels.
nWidth	Largura em pixels.
nHeight	Altura em pixels.
cCaption	Título ou conteúdo do objeto.
cTooltip	Mensagem exibida quando objeto exibe seu tooltip.
lShowHint	Flag que ativa .T. ou desativa .F. a exibição do tooltip do objeto.
cMsg	Mensagem exibida na barra de status da janela principal quando o objeto ganha foco.
nClrText	Cor do texto do objeto.
nClrPane	Cor do fundo do objeto.
bWhen	Executado quando há movimentação de foco na janela.Se retornar .T. o objeto continua habilitado, se retornar .F. o objeto será desabilitado.
bValid	Executado quando o conteúdo do objeto é modificado e deverá ser validado. Deve retornar .T. se o conteúdo é válido e .F. se conteúdo inválido.
blClicked	Executado quando acionado click do botão esquerdo do mouse sobre o objeto.
brClicked	Executado quando acionado click do botão direito do mouse sobre o objeto.
blDblClick	Executado quando acionado duplo click do botão esquerdo do mouse sobre o objeto.
oWnd	Janela onde o objeto foi criado.
lVisible	Se .T. o objeto é visível, se .F. o objeto é invisível.
Cargo	Conteúdo associado ao objeto.
bLostFocus	Executado quando objeto perde foco.
bGotFocus	Executado quando objeto ganha foco.

☑ **Construtor: Não pode ser instanciada.**

☑ **Parâmetros:**

Nenhum	-

☑ **Métodos auxiliares:**

SetFocus

- **Descrição:** Força mudança do foco de entrada de dados para o controle.
- **Sintaxe:** SetFocus( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

Hide

- **Descrição:** Torna objeto invisível.
- **Sintaxe:** Hide( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

Show

- **Descrição:** Torna objeto visível.
- **Sintaxe:** Show( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

Enable

- **Descrição:** Habilita o objeto.
- **Sintaxe: Enable**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

Disable

- **Descrição:** Desabilita o objeto.
- **Sintaxe: Disable**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

Refresh

- **Descrição:** Força atualização (sincronia) de propriedades entre o programa e o Protheus Remote.
- **Sintaxe: Refresh**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

Exemplo:

```
CLASS TCONTROL FROM TSRVOBJECT
```

TFONT()

☑ **Descrição:** Classe de objetos que define a fonte do texto utilizado nos controles visuais.

☑ **Propriedades:**

-	Herdadas as classes superiores

☑ **Construtor:** New([acName], [nPar2], [anHeight], [lPar4], [alBold], [nPar6], [lPar7], [nPar8], [alItalic], [alUnderline])

☑ **Parâmetros:**

acName	Caractere, opcional. Nome da fonte, o padrão é “Arial”.
nPar2	Reservado.
anHeight	Numérico, opcional. Tamanho da fonte. O padrão é -11.
lPar4	Reservado.
alBold	Lógico, opcional. Se .T. o estilo da fonte será negrito.
nPar6	Reservado.
lPar7	Reservado.
nPar8	Reservado.
alItalic	Lógico, opcional. Se .T. o estilo da fonte será itálico.
alUnderline	Lógico, opcional. Se .T. o estilo da fonte será sublinhado.

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg, oSay
 DEFINE MSDIALOG oDlg FROM 0,0 TO 200,200 TITLE 'My dialog' PIXEL
  // Cria font para uso
  oFont:= TFont():New('Courier New',,-14,.T.)
  // Apresenta o tSay com a fonte Courier New
  oSay := TSay():New( 10, 10, {|| 'Mensagem'},oDlg,, oFont,,,, .T.,
CLR_WHITE,CLR_RED )
  oSay:lTransparent:= .F.
 ACTIVATE MSDIALOG oDlg CENTERED
Return
```

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.

☑ **Propriedades:**

-	Herdadas as classes superiores

☑ **Construtor:** New([anTop], [anLeft], [anBottom], [anRight], [acCaption], [cPar6], [nPar7], [lPar8], [nPar9], [anClrText], [anClrBack], [oPar12], [aoWnd], [alPixel], [oPar15], [oPar16], [lPar17])

☑ **Parâmetros:**

anTop	Numérico, opcional. Coordenada vertical superior em pixels ou caracteres.
anLeft	Numérico, opcional. Coordenada horizontal esquerda em pixels ou caracteres.
anBotom	Numérico, opcional. Coordenada vertical inferior em pixels ou caracteres.
anRight	Numérico, opcional. Coordenada horizontal direita em pixels ou caracteres.
acCaption	Caractere, opcional. Título da janela.
cPar6	Reservado.
nPar7	Reservado.
lPar8	Reservado.
nPar9	Reservado.
anClrText	Numérico,opcional. Cor do texto.
anClrBack	Numérico,opcional. Cor de fundo.
oPar12	Reservado.
aoWnd	Objeto, opcional. Janela mãe da janela a ser criada, padrão é a janela principal do programa.
alPixel	Lógico, opcional. Se .T. considera as coordenadas passadas em pixels, se .F. considera caracteres.
oPar15	Reservado.
oPar16	Reservado.
nPar17	Reservado.

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
 // cria diálogo
 Local oDlg := MSDialog():New(10,10,300,300,'Meu
dialogo',,,,,CLR_BLACK,CLR_WHITE,,,.T.)

 // ativa diálogo centralizado
 oDlg:Activate(,,,.T.,{||msgstop('validou!'),.T.},,{||msgstop('iniciando…'
)} )
Return
```

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.

☑ **Propriedades:**

-	Herdadas as classes superiores

☑ **Construtor:** New([anTop], [anLeft], [anBottom], [anRight], [acCaption], [cPar6], [nPar7], [lPar8], [nPar9], [anClrText], [anClrBack], [oPar12], [aoWnd], [alPixel], [oPar15], [oPar16], [nPar17], [anWidth], [anHeight])

☑ **Parâmetros:**

anTop	Numérico, opcional. Coordenada vertical superior em pixels ou caracteres.
anLeft	Numérico, opcional. Coordenada horizontal esquerda em pixels ou caracteres.
anBotom	Numérico, opcional. Coordenada vertical inferior em pixels ou caracteres.
anRight	Numérico, opcional. Coordenada horizontal direita em pixels

	ou caracteres.
acCaption	Caractere, opcional. Título da janela.
cPar6	Reservado.
nPar7	Reservado.
lPar8	Reservado.
nPar9	Reservado.
anClrText	Numérico,opcional. Cor do texto.
anClrBack	Numérico,opcional. Cor de fundo.
oPar12	Reservado.
aoWnd	Objeto, opcional. Janela mãe da janela a ser criada, padrão é a janela principal do programa.
alPixel	Lógico, opcional. Se .T. considera as coordenadas passadas em pixels, se .F. considera caracteres.
oPar15	Reservado.
oPar16	Reservado.
nPar17	Reservado.
anWidth	Numérico, opcional. Largura da janela em pixels.
anHeight	Numérico, opcional. Altura da janela em pixels.

☑ **Métodos auxiliares:**

Activate

- **Descrição:** Ativa (exibe) o diálogo. Chamar somente uma vez este método.
- **Sintaxe:** Activate([bPar1], [bPar2], [bPar3], [alCentered], [abValid], [lPar6], [abInit], [bPar8], [bPar9])
- **Parâmetros:**

bPar1	Reservado.
bPar2	Reservado.
bPar3	Reservado.
alCentered	Lógico, opcional. Se .T. exibe a janela centralizada, .F. é padrão.
abValid	Bloco de código, opcional. Deve retornar .T. se conteúdo do diálogo é válido, se retornar .F. o diálogo não fechará quando solicitada de encerrar.
lPar6	Reservado.
abInit	Bloco de código, opcional. Executado quando o diálogo inicia exibição.
bPar8	Reservado.
bPar9	Reservado.

- **Retorno:**

Nenhum	-

End

- **Descrição:** Encerra (fecha) o diálogo.
- **Sintaxe:** End( )
- **Parâmetros:**

Nenhum	-

- **Retorno:** Lógico .T. se o diálogo foi encerrado.

Lógico	Indica se o diálogo foi encerrado.

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
 // cria diálogo
 oDlg := MSDialog():New(10,10,300,300,'Meu
dialogo',,,,,CLR_BLACK,CLR_WHITE,,,.T.)
 // ativa diálogo centralizado
 oDlg:Activate(,,,.T.,{||msgstop('validou!'),.T.},,{||msgstop('iniciando..
.')} )

 /* os comandos abaixo proporcionam o mesmo resultado
 // cria diálogo
 DEFINE DIALOG oDlg TITLE 'Meu dialogo' FROM 10,10 TO 300,300 COLOR
CLR_BLACK,CLR_WHITE PIXEL
 // ativa diálogo centralizado
 ACTIVATE DIALOG oDlg CENTER ON INIT (msgstop('iniciando...')) VALID
(msgstop('validou!'),.T.)
 */
Return Nil
```

TWINDOW()

☑ **Descrição:** Classe de objetos do tipo diálogo principal de programa. Deverá existir apenas uma instância deste objeto na execução do programa.

☑ **Propriedades:**

-	Herdadas das classes superiores

☑ **Construtor:** New([anTop], [anLeft],[anBottom], [anRight], [acTitle], [nPar6], [oPar7] ,[oPar8],[oPar9], [aoParent], [lPar11], [lPar12], [anClrFore], [anClrBack], [oPar15], [cPar16], [lPar17], [lPar18], [lPar19], [lPar20], [alPixel]);

☑ **Parâmetros:**

nTop	Numérico, opcional. Coordenada vertical superior em pixels ou caracteres.
nLeft	Numérico, opcional. Coordenada horizontal esquerda em pixels ou caracteres.
nBottom	Numérico, opcional. Coordenada vertical inferior em pixels ou caracteres.
nRight	Numérico, opcional. Coordenada horizontal inferior em pixels ou caracteres.
cTitle	Caractere, opcional. Título da janela.
nPar6	Reservado.
oPar7	Reservado.
oPar8	Reservado.
oPar9	Reservado.
oParent	Objeto, opcional. Janela mãe da janela corrente.
lPar11	Reservado.
lPar12	Reservado.
nClrFore	Numérico, opcional. Cor de fundo da janela.
nClrText	Numérico, opcional. Cor do texto da janela.
oPar15	Reservado.
cPar16	Reservado.
lPar17	Reservado.
lPar18	Reservado.
lPar19	Reservado.
lPar20	Reservado.
lPixel	Lógico, opcional. Se .T. (padrão) considera coordenadas passadas em pixels, se .F. considera caracteres.

☑ **Métodos auxiliares:**

Activate

- **Descrição:** Ativa (exibe) a janela. Chamar esse método apenas uma vez.
- **Sintaxe:** Activate([acShow], [bPar2], [bPar3], [bPar4], [bPar5], [bPar6], [abInit], [bPar8], [bPar9], [bPar10], [bPar11], [bPar12] ,[bPar13], [bPar14], [bPar15], [abValid], [bPar17], [bPar18]).
- **Parâmetros:**

acShow	Caracter, opcional. "ICONIZED" para janela iconizada ou "MAXIMIZED" para janela maximizada.
bPar2	Reservado.
bPar3	Reservado.
bPar4	Reservado.
bPar5	Reservado.
bPar6	Reservado.
abInit	Bloco de código. Executado quando janela está sendo exibida.
bPar8	Reservado.
bPar9	Reservado.
bPar10	Reservado.
bPar11	Reservado.
bPar12	Reservado.
bPar13	Reservado.
bPar14	Reservado.
bPar15	Reservado.
abValid	Bloco de código. Executado quando a janela for solicitada de fechar. Deverá retornar .T. se o conteúdo da janela for válido, ou .F. se não. Se o bloco retornar .F. a janela não fechará.
bPar17	Reservado.
bPar18	Reservado.

- **Retorno:**

Nenhum	-

End

- **Descrição:** Solicita encerramento da janela.
- **Sintaxe:** End( )
- **Parâmetros:**

Nenhum	-

- **Retorno:** Lógico .T. se o diálogo foi encerrado.

Lógico	Indica se a janela foi encerrada.

Center

- **Descrição:** Centraliza a janela.
- **Sintaxe:** Center( )
- **Parâmetros:**

Nenhum	-

- **Retorno:** Lógico .T. se o diálogo foi encerrado.

Nenhum	-

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oWindow
Local abInit:= {||conout('ativando!')}
Local abValid:= {||conout('encerrando!'),.T.}
 oWindow:= tWindow():New( 10, 10, 200, 200, 'Meu
programa',,,,,,,,CLR_WHITE,CLR_BLACK,,,,,,,.T. )
 oWindow:Activate('MAXIMIZED',,,,,,abInit,,,,,,,,,abValid,,)

 /* os comandos abaixo proporcionam o mesmo resultado
 DEFINE WINDOW oWindow FROM 10, 10 TO 200,200 PIXEL TITLE 'Meu programa'
COLOR CLR_WHITE,CLR_BLACK
 ACTIVATE WINDOW oWindow MAXIMIZED ON INIT abInit VALID abValid
 */
Return
.
```

TCONTROL()

☑ **Descrição:** Classe abstrata comum entre todos os componentes visuais editáveis. Não deve ser instanciada diretamente.

☑ **Propriedades:**

Align	Numérico. Alinhamento do controle no espaço disponibilizado pelo seu objeto parente. 0 = Nenhum (padrão), 1= no topo, 2 = no rodapé, 3= a esquerda, 4 = a direita e 5 = em todo o parente.
lModified	Lógico. Se .T. indica que o conteúdo da variável associada ao controle foi modificado.
lReadOnly	Lógico. Se .T. o conteúdo da variável associada ao controle permanecerá apenas para leitura.
hParent	Numérico. Handle (identificador) do objeto sobre o qual o controle foi criado.
bChange	Bloco de código. Executado quando o estado ou conteúdo do controle é modificado pela ação sobre o controle.

☑ **Construtor: Não pode ser instanciada.**

☑ **Parâmetros:**

Nenhum	-

☑ **Métodos auxiliares:**

SetFocus

- **Descrição:** Força mudança do foco de entrada de dados para o controle.
- **Sintaxe:** SetFocus( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

Exemplo:

```
CLASS TSay FROM Tcontrol
```

BRGETDDB()

☑ **Descrição:** Classe de objetos visuais do tipo Grid.

☑ **Propriedades:**

+	Herdadas da classe superior
nAt	Linha atualmente selecionada / posicionada no objeto
nLen	Número total de linhas do objeto

☑ **Construtor:** New([nRow], [nCol], [nWidth], [nHeigth], [bFields], [aHeaders], [aColSizes], [oDlg], [cField], [uValue1], [uValue2], [uChange], [uLDblClick], [uRClick], [oFont], [oCursor], [nClrFore], [nClrBack], [cMsg], [lUpdate], [cAlias], [lPixel], [bWhen], [lDesign], [bValid], [aAlter], [oMother])

☑ **Parâmetros:**

nRow	Numérico, opcional. Coordenada vertical
nCol	Numérico, opcional. Coordenada horizontal
nWidth	Numérico, opcional. Largura do objeto
nHeight	Numérico, opcional. Altura do objeto
bFields	Bloco de código, Lista de Campos
aHeaders	Vetor, Descrição dos campos para no cabeçalho
aColSizes	Vetor, Largura das colunas
oDlg	Objeto, opcional. Janela ou controle onde o divisor deverá ser criado.
cField	Caracter, opcional. Campos necessários ao filtro
uValue1	Indefinido, opcional. Inicio do intervalo para o filtro
uValue2	Indefinido, opcional. Fim do intervalo para o filtro
bChange	Bloco de código, opcional. Executado quando o item selecionado é alterado.
bLDblClick	Bloco de código, opcional. Executado quando acionado duplo click do botão esquerdo do mouse sobre o controle.
bRClick	Não utilizado
oFont	Objeto, opcional. Fonte
oCursor	Objeto, opcional. Tipo do Cursor
nClrFore	Numérico, opcional. Cor do texto da janela.
nClrBack	Numérico, opcional. Cor de fundo da janela.
cMsg	Caracter, opcional. Mesamge ao posicionar o mouse sobre o objeto
lUpdate	Não utilizado
cAlias	Caracter, opcional se objeto utilizado com Vetor, obrigatorio se utilizado com Tabela
lPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
bWhen	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.
lDesign	Não Utilizado

bValid	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.
aAlter	Não Utilizado
oMother	Não Utilizado

☑ **Métodos auxiliares:**

GoUp

- **Descrição:** Salta uma linha para cima.
- **Sintaxe: GoUp**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoDown

- **Descrição:** Salta uma linha para baixo.
- **Sintaxe: GoDown**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoTop

- **Descrição:** Salta para primeira linha.
- **Sintaxe: GoTop**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoBottom

- **Descrição:** Salta para ultima linha.
- **Sintaxe: GoBottom**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

RowCount

- **Descrição:** Retorna numero de linhas visiveis.
- **Sintaxe: RowCount**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 302,402 PIXEL TITLE 'Exemplo'

      DbSelectArea('SA1')
      oBrowse := BrGetDDB():New(
1,1,200,150,,,,oDlg,,,,,,,,,,,.F.,'SA1',.T.,,.F.,,, )
      oBrowse:AddColumn(TCColumn():New('Codigo',{||SA1->A1_COD
},,,,'LEFT',,.F.,.F.,,,,.F.,))
      oBrowse:AddColumn(TCColumn():New('Loja'  ,{||SA1-
>A1_LOJA},,,,'LEFT',,.F.,.F.,,,,.F.,))
      oBrowse:AddColumn(TCColumn():New('Nome'  ,{||SA1-
>A1_NOME},,,,'LEFT',,.F.,.F.,,,,.F.,))

      // Principais commandos
      TButton():New(160,001,'GoUp()',oDlg,{|| oBrowse:GoUp()
},40,10,,,,.T.)
      TButton():New(170,001,'GoDown()',oDlg,{|| oBrowse:GoDown()
},40,10,,,,.T.)
      TButton():New(180,001,'GoTop()',oDlg,{|| oBrowse:GoTop()
},40,10,,,,.T.)
      TButton():New(190,001,'GoBottom()' , oDlg,{|| oBrowse:GoBottom()
},40,10,,,,.T.)
      TButton():New(160,060,'nAt (Linha selecionada)',oDlg,;
                          {|| Alert (oBrowse:nAt)},80,10,,,,.T.)
      TButton():New(170,060,'nRowCount (Nr de linhas visiveis)',oDlg,;
                          {|| Alert(oBrowse:nRowCount()) },80,10,,,,.T.)
      TButton():New(180,060, 'nLen (Numero total de linhas)', oDlg,;
                          {|| Alert(oBrowse:nLen) },80,10,,,,.T.)


      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

MSCALEND()

☑ **Descrição:** Classe de objetos visuais do tipo Calendário.

☑ **Propriedades:**

+	Herdadas da classe superior
bChange	Bloco de código executado na mudança de seleção de um dia.
bChangeMes	Bloco de código executado na mundaça de seleção de um mês.
dDiaAtu	Dia atualmente selecionado

☑ **Construtor:** New([nRow], [nCol], [oDlg], [lCanMultSel])

☑ **Parâmetros:**

nRow	Numérico, opcional. Coordenada vertical	
nCol	Numérico, opcional. Coordenada horizontal	
oDlg	Objeto, opcional. Janela ou controle onde o divisor deverá ser criado.	
lCanMultSel	Logico, opcional. Permite seleção de multiplos dias	

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
user function Calend_Ex()
      DEFINE MSDIALOG oDlg FROM 0,0 TO 1000,1000 PIXEL TITLE 'Exemplo de
MsCalend'
            // Cria objeto
            oCalend:=MsCalend():New(01,01,oDlg,.T.)
            // Code-Block para mudança de Dia
            oCalend:bChange := {|| Alert('Dia Selecionado: ' +
dtoc(oCalend:dDiaAtu)) }
            // Code-Block para mudança de mes
            oCalend:bChangeMes := {|| alert('Mes alterado') }
      ACTIVATE MSDIALOG oDlg CENTERED
Return
```

MSCALENDGRID()

☑ **Descrição:** Classe de objetos visuais do tipo Grade de Períodos.

☑ **Propriedades:**

bAction	Bloco de código. Executado quando o botão esquerdo do mouse é pressionado.
bRClick	Bloco de código. Executado quando o botão direito do mouse é pressionado.

☑ **Construtor:** New([aoWnd], [anRow], [anCol], [anHeight], [anWidth], [dDateIni], [nResolution], [abWhen], [abAction], [nDefColor], [bRClick], [lFillAll])

☑ **Parâmetros:**

aoWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
anRow	Numérico, opcional. Coordenada vertical em pixels ou carateres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
anHeight	Numérico, opcional. Altura do botão em pixels.
anWidth	Numérico, opcional. Largura do botão em pixels.
dDateIni	Data, Data inicial do Calendário
nResolution	Numérico, Resolução a ser aplicada na grid do Calendário
abWhen	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.
abAction	Bloco de código, opcional. Bloco que deverá ser acionado quando o botão for pressionado.
nDefColor	Numérico, opcional. Cor do Fundo da Grid.
bRClick	Bloco de código, opcional. Executado quando acionado o botão direito do mouse sobre o controle.
lFillAll	Lógico, opcional. Preenche todo o periodo

☑ **Métodos auxiliares:**

Add

- **Descrição:** Adiciona periodo na Grid.
- **Sintaxe:** Add(cCaption, nLin, nInitial, nFinal, nColor, cDescri)
- **Parâmetros:**

cCaption	Caracter. Descrição a ser inserida a esquerda da Grid
nLin	Numerico. Numero da linha a ser inserido o item
nInitial	Numerico. Dia inicial
nFinal	Numerico. Dia Final

nColor	Numerico. Cor que destacara o item
cDescri	Caracter, Opicional. Descrição do tipo tooltip(hint) do item

❑ **Retorno:**

Nil	

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 304,504 PIXEL TITLE 'Exemplo'
            nResolution := 1
            oMsCalendGrid := MsCalendGrid():New( oDlg, 01, 01, 500,300,;
                                    date(), nResolution, ,{|x,y| Alert(x) },;
                                    RGB(255,255,196), {|x,y|Alert(x,y)}, .T.
)
            oMsCalendGrid:Add('caption 01', 1, 10, 20, RGB(255,000,0),
'Descricao 01')
            oMsCalendGrid:Add('caption 02', 2, 20, 30, RGB(255,255,0),
'Descricao 02')
            oMsCalendGrid:Add('caption 03', 3, 01, 05, RGB(255,0,255),
'Descricao 03')
      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

MSSELBR()

☑ **Descrição:** Classe de objetos visuais do tipo controle - Grid

☑ **Propriedades:**

+	Herdadas da classe superior
nAt	Linha atualmente selecionada / posicionada no objeto
nLen	Número total de linhas do objeto

☑ **Construtor:** New([nRow], [nCol], [nWidth], [nHeigth], [bFields], [aHeaders], [aColSizes], [oDlg], [cField], [uValue1], [uValue2], [uChange], [uLDblClick], [uRClick], [oFont], [oCursor], [nClrFore], [nClrBack], [cMsg], [lUpdate], [cAlias], [lPixel], [bWhen], [lDesign], [bValid])

☑ **Parâmetros:**

nRow	Numérico, opcional. Coordenada vertical
nCol	Numérico, opcional. Coordenada horizontal
nWidth	Numérico, opcional. Largura do objeto
nHeight	Numérico, opcional. Altura do objeto
bFields	Bloco de código, Lista de Campos
aHeaders	Vetor, Descrição dos campos para no cabeçalho
aColSizes	Vetor, Largura das colunas
oDlg	Objeto, opcional. Janela ou controle onde o divisor deverá ser criado.
cField	Caracter, opcional. Campos necessários ao filtro
uValue1	Indefinido, opcional. Inicio do intervalo para o filtro
uValue2	Indefinido, opcional. Fim do intervalo para o filtro
bChange	Bloco de código, opcional. Executado quando o item selecionado é alterado.
bLDblClick	Bloco de código, opcional. Executado quando acionado duplo click do botão esquerdo do mouse sobre o controle.
bRClick	Não utilizado
oFont	Objeto, opcional. Fonte
oCursor	Objeto, opcional. Tipo do Cursor
nClrFore	Numérico, opcional. Cor do texto da janela.
nClrBack	Numérico, opcional. Cor de fundo da janela.
cMsg	Caracter, opcional. Mesamge ao posicionar o mouse sobre o objeto
lUpdate	Não utilizado
cAlias	Caracter, opcional se objeto utilizado com Vetor, obrigatorio se utilizado com Tabela
lPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
bWhen	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.
lDesign	Não Utilizado
bValid	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.

☑ **Métodos auxiliares:**

GoUp

- **Descrição:** Salta uma linha para cima.
- **Sintaxe: GoUp**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoDown

- **Descrição:** Salta uma linha para baixo.
- **Sintaxe: GoDown**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoTop

- **Descrição:** Salta para primeira linha.
- **Sintaxe: GoTop**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoBottom

- **Descrição:** Salta para ultima linha.
- **Sintaxe: GoBottom**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

RowCount

- **Descrição:** Retorna numero de linhas visiveis.
- **Sintaxe: RowCount**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 302,402 PIXEL TITLE 'Exemplo'

      DbSelectArea('SA1')
      oBrowse := MsSelBr():New(
1,1,200,150,,,,oDlg,,,,,,,,,,,.F.,'SA1',.T.,,.F.,,, )
      oBrowse:AddColumn(TCColumn():New('Codigo',{||SA1->A1_COD
},,,,'LEFT',,.F.,.F.,,,,.F.,))
      oBrowse:AddColumn(TCColumn():New('Loja'  ,{||SA1-
>A1_LOJA},,,,'LEFT',,.F.,.F.,,,,.F.,))
      oBrowse:AddColumn(TCColumn():New('Nome'  ,{||SA1-
>A1_NOME},,,,'LEFT',,.F.,.F.,,,,.F.,))

      // Principais commandos
      TButton():New(160,001,'GoUp()',oDlg,{|| oBrowse:GoUp()
},40,10,,,,.T.)
      TButton():New(170,001,'GoDown()',oDlg,{|| oBrowse:GoDown()
},40,10,,,,.T.)
      TButton():New(180,001,'GoTop()',oDlg,{|| oBrowse:GoTop()
},40,10,,,,.T.)
      TButton():New(190,001,'GoBottom()' , oDlg,{|| oBrowse:GoBottom()
},40,10,,,,.T.)
      TButton():New(160,060,'nAt (Linha selecionada)',oDlg,{||
Alert(oBrowse:nAt)},80,10,,,,.T.)
      TButton():New(170,060,'nRowCount (Nr de linhas visiveis)',oDlg,;
                          {|| Alert(oBrowse:nRowCount()) },80,10,,,,.T.)
      TButton():New(180,060, 'nLen (Numero total de linhas)', oDlg,;
                          {|| Alert(oBrowse:nLen) },80,10,,,,.T.)


      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
.
```

MSWORKTIME()

☑ **Descrição:** Classe de objetos visuais do tipo controle - Barra de Período.

☑ **Propriedades:**

bChange	Bloco de código. Executado ao disparar qualquer ação sobre o objeto.

☑ **Construtor:** New([aoWnd], [anRow], [anCol], [anHeight], [anWidth], [nResolution], [cValue], [abWhen], [abChange])

☑ **Parâmetros:**

aoWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
anRow	Numérico, opcional. Coordenada vertical em pixels ou carateres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou

	caracteres.
anHeight	Numérico, opcional. Altura do botão em pixels.
anWidth	Numérico, opcional. Largura do botão em pixels.
nResolution	Numérico, Resolução a ser aplicada na grid do Calendário
cValue	Caracter, opcional. Descritivo
abWhen	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.
abChange	Bloco de código, opcional. Bloco que deverá ser acionado quando o botão for pressionado.

☑ **Métodos auxiliares:**

GetValue

- **Descrição:** Retorna os item selecionados no formato "XX X XX".
- **Sintaxe:** GetValue()
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Caracter	Retorna os item selecionados no formato "XX X XX".

GetInterTime

- **Descrição:** Retorna periodo selecionado.
- **Sintaxe:** GetInterTime()
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Caracter	Retorna o período selecionada no formato "HH:MM:SS"

SetResol

- **Descrição:** Define a resolução da demonstração da barra de períodos.
- **Sintaxe:** SetResol (nResolution)
- **Parâmetros:**

nResolution	Resolução

- **Retorno:**

Nil	

SetValue

- **Descrição:** Atribui um determindo conteúdo para um objeto ou atributo de um objeto.
- **Sintaxe:** SetValue(xValue, cPicture)
- **Parâmetros:**

xValue	Valor que será atribuído ao objeto ou atributo do objeto.
cPicture	Picture de formação opcional para o conteúdo atribuído.

- **Retorno:**

Nenhum	

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 304,504 PIXEL TITLE 'Exemplo'
      oMsWorkTime := MsWorkTime():New(oDlg,04,04,280,133,0,'',{||.T.},;
                    {|oWorkTime|Alert('GetValue():
'+oWorkTime:GetValue()+chr(13)+;
                                  'GetInterTime():
'+oWorkTime:GetInterTime())} )
       oMsWorkTime:SetValue('X X XX X')
      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

SBUTTON()

☑ **Descrição:** Classe de objetos visuais do tipo botão que 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.

☑ **Propriedades:**

bAction	Bloco de código. Executado ao precionar o botão esquerdo do mouse.

☑ **Construtor:** New([nTop], [nLeft], [nType], [bAction], [oWnd], [lEnable], [cMsg], [bWhen])

☑ **Parâmetros:**

Parâmetro	Tipo / Descrição
nTop	Numérico, opcional. Coordenada vertical em pixels ou carateres.
nLeft	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
nType	Numérico. Tipo do Botão
bAction	Bloco de código, opcional. Bloco que deverá ser acionado quando o botão for pressionado.
oWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
lEnable	Logico, opcional. Habilita/Desabilita botão
cMsg	Caracter, Opicional. Descrição do tipo tooltip(hint) do item
abWhen	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.

☑ **Métodos auxiliares:**

Create

- **Descrição:** Retorna Método construtor opcional da classe.
- **Sintaxe:** Create()
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Objeto	Objeto da classe Sbutton com todos os atributos com conteúdos padrões.

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 304,504 PIXEL TITLE 'Exemplo'
          oSButton1 := SButton():New( 01,01,1,{||Alert('SButton
01')},oDlg,.T.,,)
          oSButton2 := SButton():New( 21,01,2,{||Alert('SButton
02')},oDlg,.T.,,)
          oSButton3 := SButton():New( 41,01,3,{||Alert('SButton
03')},oDlg,.T.,,)
          oSButton4 := SButton():New( 61,01,4,{||Alert('SButton
04')},oDlg,.T.,,)
      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

Anotações

TBAR()

☑ **Descrição:** 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.

☑ **Propriedades:**

-	Herdados das classes superiores.

☑ **Construtor:**

☑ **Parâmetros:**

oWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
nBtnWidth	Numérico, opcional. Largura do botão contido na barra
nBtnHeight	Numérico, opcional. Altura do botão contido na barra
l3D	Lógico, opcional. Define tipo da barra
cMode	Não utilizado.
oCursor	Objeto, opcional. Define Cursor ao posicionar o mouse sobre a barra.
cResource	Caracter, opcional. Imagem do recurso a ser inserido como fundo da barra.
lNoAutoAdjust	Lógico.

☑ **Aparência:**

Exemplo:

```
#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:lCentered := .T.
oDlg:Activate()

Return
```

Anotações

TBITMAP()

☑ **Descrição:** Classe de objetos visuais que permite a exibição de uma imagem.

☑ **Propriedades:**

cResName	Caractere, Nome do resource
cBmpFile	Caractere, Nome do arquivo
lStretch	Lógico, opcional. Strech da imagem
lAutoSize	Lógico, opcional. Tamanho automático
lTransparent	Lógico, opcional. Transparente

☑ **Construtor:** New([anTop], [anLeft], [anWidth], [anHeight], [acResName], [acBmpFile], [alNoBorder], [aoWnd], [abLClicked], [abRClicked], [alScroll], [alStretch], [aoCursor], [acMsg], [alUpdate], [abWhen], [alPixel], [abValid], [alDesign], [alIsIcon], [alIsJpeg])

☑ **Parâmetros:**

anTop	Numérico, opcional. Coordenada vertical em pixels ou carateres.
anLeft	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
anWidth	Numérico, opcional. Largura em pixels.
anHeight	Numérico, opcional. Altura em pixels.
acResName	Caractere, Nome do resource.
acBmpFile	Caractere, Nome do arquivo.
alNoBorder	Lógico, opcional. Exibe borda
aoWnd	Objeto, opcional. Janela ou controle onde a imagem deverá ser criada.
abLClicked	Bloco de código, opcional. Bloco que deverá ser acionado botão esquerdo do mouse for pressionado.
abRClicked	Bloco de código, opcional. Bloco que deverá ser acionado botão direito do mouse for pressionado.
alScroll	Lógico, opcional. Ativa o scroll
alStretch	Lógico, opcional. Strech da imagem
aoCursor	Objeto, opcional. Cursor a ser exibido
acMsg	Não utilizado
alUpdate	Não utilizado
abWhen	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.
alPixel	Lógico, opcional. Utiliza coordenadas em pixel
abValid	Bloco de código, opcional. Bloco que deverá ser acionado na validação.
alDesign	Não utilizado
alIsIcon	Lógico, opcional. Ícone. Não utilizado a partir da versão Protheus 8
alIsJpeg	Lógico, opcional. Jpeg. Não utilizado a partir da versão Protheus 8

☑ **Métodos auxiliares:**

Create

- **Descrição:** Retorna Método construtor opcional da classe.
- **Sintaxe:** Create()
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Objeto	Objeto da classe TBar com todos os atributos com conteúdos padrões.

SetBmp

- **Descrição:** Método para carregar uma imagem do repositório.
- **Sintaxe:** SetBmp(acResName)
- **Parâmetros:**

acResName	Caractere, Nome do resource

- **Retorno:**

Nenhum	-

Load

- **Descrição:** Método para carregar uma imagem do repositório ou local.
- **Sintaxe:** Load(acResName, acBmpFile)
- **Parâmetros:**

acResName	Caractere, Nome do resource.
acBmpFile	Caractere, Nome do arquivo.

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
#include “protheus.ch”


User Function tBitmapTst()
Local oDlg
Local oBmp

DEFINE MSDIALOG oDlg TITLE '' FROM 0,0 TO 280,330 PIXEL

@ 10, 10 BITMAP oBmp RESOURCE 'totvs.bmp' SIZE 150,150 OF oDlg PIXEL
oBmp:lAutoSize := .T.

ACTIVATE MSDIALOG oDlg CENTERED

Return

.
```

TBROWSEBUTTON()

☑ **Descrição:** Classe de objetos visuais do tipo botão no formato padrão utilizado em browses da aplicação.

☑ **Propriedades:**

bAction	Bloco de código. Executado ao precionar o botão esquerdo do mouse.

☑ **Construtor:** New([nRow], [nCol], [cCaption], [oWnd], [bAction], [nWidth], [nHeight], [nHelpId], [oFont], [lDefault], [lPixel], [lDesign], [cMsg], [lUpdate], [bWhen], [bValid], [lCancel])

☑ **Parâmetros:**

nRow	Numérico, opcional. Coordenada vertical em pixels ou carateres.
nCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
cCaption	Caractere, opcional. Titulo do botão.
oWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
bAction	Bloco de código, opcional. Bloco que deverá ser acionado quando o botão for pressionado.
nWidth	Numérico, opcional. Largura do botão em pixels
nHeight	Numérico, opcional. Altura do botão em pixels.
nHelpId	Reservado
oFont	Objeto, opcional. Objeto tipo tFont com propriedades da fonte utilizada para o título do botão.
lDefault	Reservado
lPixel	Lógico, opcional. Se .T. considera as coordenadas passadas em pixels, se .F. (padrão) considera em caracteres.

lDesign	Reservado
cMsg	Reservado
lUpdate	Reservado
bWhen	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.
bValid	Reservado
lCancel	Reservado

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 304,504 PIXEL TITLE 'Exemplo'
      oTBrowseButton := TBrowseButton():New( 01,01,'TBrowseButton',oDlg,;

{||Alert('TBrowseButton')},40,10,,,.F.,.T.,.F.,,.F.,,,)

      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

Anotações

TBTNBMP()

☑ **Descrição:** Classe de objetos visuais do tipo botão, o qual permite que seja vinculada uma imagem ao controle.

☑ **Propriedades:**

bAction	Bloco de código. Executado ao precionar o botão esquerdo do mouse.

☑ **Construtor:** NewBar([cResName1], [cResName2], [cBmpFile1], [cBmpFile2], [cMsg], [bAction], [lGroup], [oWnd], [lAdjust], [bWhen], [cToolTip], [lPressed], [bDrop], [cAction], [nPos], [cPrompt], [oFont], [cResName3], [cBmpFile3], [lBorder])

☑ **Parâmetros:**

cResName1	Caractere, Nome do resource
cResName2	Caractere, Nome do resource
cBmpFile1	Caractere, NÃO UTILIZADO
cBmpFile2	Caractere, NÃO UTILIZADO
cMsg	Caractere, Mensagem de Hint
bAction	Bloco de código. Ação executada
lGroup	Lógico. Define grupo
oWnd	Objeto, opcional. Janela ou controle onde a botão deverá ser criado
lAdjust	Lógico, NÃO UTILIZADO
bWhen	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.
cToolTip	Mensagem exibida quando objeto exibe seu tooltip.
lPressed	Não utilizado
bDrop	Não utilizado
cAction	Não utilizado
nPos	Não utilizado
cPrompt	Caracter, opcional. Caption do botão.
oFont	Objeto, opcional. Objeto tipo tFont utilizado para definir as características da fonte utilizada para exibir o conteúdo do controle.
cResName3	Não utilizado
cBmpFile3	Não utilizado
lBorder	Não utilizado

☑ **Métodos auxiliares:**

LoadBitmaps

- **Descrição:** Atribui ao botão os bitmaps ou recursos para exibição.
- **Sintaxe:** LoadBitmaps([cResName1], [cResName2], [cBmpFile1], [cBmpFile2], [cResName3], [cBmpFile3])

- **Parâmetros:**

cResName1	Caractere, Nome do resource
cResName2	Caractere, Nome do resource
cBmpFile1	Caractere, Nome do arquivo BMP
cBmpFile2	Caractere, Nome do arquivo BMP
cResName3	Caractere, Nome do resource
cBmpFile3	Caractere, Nome do resource

- **Retorno:**

Nenhum	

SetPopUpMenu

- **Descrição:** Seta o objeto do tipo TMenu referente ao botão.
- **Sintaxe:** SetPopupMenu(oMenu)
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 304,504 PIXEL TITLE 'Exemplo'
      oTBar := TBar():New( oDlg,25,32,.T.,,,,.F. ) // Orig: 25,45
      oTBtnBmp1 := TBtnBmp() :NewBar('RPMNEW',,,,'Msg 01',;
                    {||Alert('TBtnBmp
01')},.F.,oTBar,.T.,{||.T.},,.F.,,,1,,,,,.T. )
      oTBtnBmp2 := TBtnBmp() :NewBar('copyuser',,,,'Msg 02',;
                      {||Alert('TBtnBmp
02')},.F.,oTBar,.T.,{||.T.},,.F.,,,1,,,,,.T. )

      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

TBTNBMP2()

☑ **Descrição:** Classe de objetos visuais do tipo botão, o qual permite a exibição de uma imagem ou de um popup.

☑ **Propriedades:**

-	Herdadas das classes superiores.

☑ **Construtor:** New([anTop], [anLeft], [anWidth], [anHeight], [acResName1], [acResName2], [acBmpFile1], [acBmpFile2],[abAction], [aoWnd], [acMsg], [abWhen], [lAdjuste], [lUpdate])

☑ **Parâmetros:**

anTop	Numérico, opcional. Coordenada vertical em pixels ou carateres.
anLeft	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
anWidth	Numérico, opcional. Largura em pixels.
anHeight	Numérico, opcional. Altura em pixels.
acResName1	Caractere, Nome do resource
acResName2	Caractere, NÃO UTILIZADO
acBmpFile1	Caractere, NÃO UTILIZADO
acBmpFile2	Caractere, NÃO UTILIZADO
abAction	Bloco de código. Ação executada
aoWnd	Objeto, opcional. Janela ou controle onde a botão deverá ser criado
acMsg	Caractere, Mensagem de Hint
abWhen	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.
lAdjust	Lógico, NÃO UTILIZADO
lUpdate	Lógico, NÃO UTILIZADO

☑ **Métodos auxiliares:**

LoadBitmaps

- **Descrição:** Atribui ao botão os bitmaps ou recursos para exibição.
- **Sintaxe:** LoadBitmaps([cResName1], [cResName2], [cBmpFile1], [cBmpFile2], [cResName3], [cBmpFile3])

- **Parâmetros:**

cResName1	Caractere, Nome do resource
cResName2	Caractere, Nome do resource
cBmpFile1	Caractere, Nome do arquivo BMP
cBmpFile2	Caractere, Nome do arquivo BMP
cResName3	Caractere, Nome do resource
cBmpFile3	Caractere, Nome do resource

- **Retorno:**

Nenhum	

SetPopUpMenu

- **Descrição:** Seta o objeto do tipo TMenu referente ao botão.
- **Sintaxe:** SetPopupMenu(oMenu)
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
Local oBtn := TBtnBmp2():New( 10, 10, 25, 25, 'printer_r2' , , , , , oDlg,
, , )
Exemplo 2 (Popup)
#INCLUDE 'PROTHEUS.CH'
User Function BtnBmpPopUp
Local oDlg
Local oBtn
DEFINE MSDIALOG oDlg TITLE '' FROM 0,0 TO 100,200 PIXEL
oBtn := TBtnBmp2():New( 10, 10, 13, 13, 'IBF-MENU.BMP' , 'IBF-MENU.BMP' ,
, , , oDlg, , , .T.)
oBtn:SetPopupmenu(TShowMenu())
ACTIVATE MSDIALOG oDlg CENTERED
Return

/*************************/
Static Function TShowMenu()
Local oMenu
oMenu := TMenu():New(0,0,0,0,.T.)
oMenu:Add(TMenuItem():New(,'Detalhes', 'Detalhes',,, ,,,,,,,,,.T.))
oMenu:Add(TMenuItem():New(,'Add Info', 'Add Info',,, ,,,,,,,,,.T.))
Return oMenu
```

TBUTTON()

☑ **Descrição:** Classe de objetos visuais do tipo botão, o qual permite a utilização de texto para sua identificação.

☑ **Propriedades:**

lProcessing	Lógico. Se .T. indica o botão está efetuando uma ação.
bAction	Bloco de código. Executado quando o botão é pressionado.

☑ **Construtor:** New([anRow], [anCol], [acCaption], [aoWnd], [abAction], [anWidth], [anHeight], [nPar8], [aoFont], [lPar10], [alPixel],[lPar12],[cPar13], [lPar14], [abWhen], [bPar16], [lPar17])

☑ **Parâmetros:**

Parâmetro	Tipo / Descrição
anRow	Numérico, opcional. Coordenada vertical em pixels ou carateres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
acCaption	Caractere, opcional. Titulo do botão.
aoWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
abAction	Bloco de código, opcional. Bloco que deverá ser acionado quando o botão for pressionado.
anWidth	Numérico, opcional. Largura do botão em pixels.
anHeight	Numérico, opcional. Altura do botão em pixels.
nPar8	Reservado.
aoFont	Objeto, opcional. Objeto tipo tFont com propriedades da fonte utilizada para o título do botão.

lPar10	Reservado.
alPixel	Lógico, opcional. Se .T. considera as coordenadas passadas em pixels, se .F. (padrão) considera em caracteres.
lPar12	Reservado.
cPar13	Reservado.
lPar14	Reservado.
abWhen	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.
bPar16	Reservado.
lPar17	Reservado.

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function TesteGet()
Local oDlg, oButton, oCombo, cCombo
aItems:= {'item1','item2','item3'}
cCombo:= aItems[2]
DEFINE MSDIALOG oDlg FROM 0,0 TO 300,300 PIXEL TITLE 'Meu Combo'
 oCombo:= tComboBox():New(10,10,{|u|if(PCount()>0,cCombo:=u,cCombo)},;
 aItems,100,20,oDlg,,{||MsgStop('Mudou item')},,,,.T.,,,,,,,,,'cCombo')

 // Botão para fechar a janela
 oButton:=tButton():New(30,10,'fechar',oDlg,{||oDlg:End()},100,20,,,,.T.)
ACTIVATE MSDIALOG oDlg CENTERED
MsgStop( 'O valor é '+cCombo )
Return NIL
```

TCBROWSE()

☑ **Descrição:** Classe de objetos visuais do tipo Grid.

☑ **Propriedades:**

+	Herdadas da classe superior
nAt	Linha atualmente selecionada / posicionada no objeto
nLen	Número total de linhas do objeto

☑ **Construtor:** New([nRow], [nCol], [nWidth], [nHeigth],[bFields], [aHeaders], [aColSizes], [oDlg], [cField], [uValue1], [uValue2], [uChange],[{|nRow,nCol,nFlags|[uLDblClick]}], [{|nRow,nCol,nFlags|[uRClick]}], [oFont], [oCursor], [nClrFore], [nClrBack], [cMsg], [lUpdate], [cAlias], [lPixel], [{uWhen}], [lDesign], [bValid], [lHScroll], [lVScroll])

☑ **Parâmetros:**

nRow	Numérico, opcional. Coordenada vertical
nCol	Numérico, opcional. Coordenada horizontal
nWidth	Numérico, opcional. Largura do objeto
nHeight	Numérico, opcional. Altura do objeto
bFields	Bloco de código, Lista de Campos
aHeaders	Vetor, Descrição dos campos para no cabeçalho
aColSizes	Vetor, Largura das colunas
oDlg	Objeto, opcional. Janela ou controle onde o divisor deverá ser criado.
cField	Caracter, opcional. Campos necessários ao filtro
uValue1	Indefinido, opcional. Inicio do intervalo para o filtro
uValue2	Indefinido, opcional. Fim do intervalo para o filtro
bChange	Bloco de código, opcional. Executado quando o item selecionado é alterado.
bLDblClick	Bloco de código, opcional. Executado quando acionado duplo click do botão esquerdo do mouse sobre o controle.
bRClick	Não utilizado
oFont	Objeto, opcional. Fonte
oCursor	Objeto, opcional. Tipo do Cursor
nClrFore	Numérico, opcional. Cor do texto da janela.
nClrBack	Numérico, opcional. Cor de fundo da janela.
cMsg	Caracter, opcional. Mesamge ao posicionar o mouse sobre o objeto
lUpdate	Não utilizado
cAlias	Caracter, opcional se objeto utilizado com Vetor, obrigatorio se utilizado com Tabela
lPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
bWhen	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.
lDesign	Não Utilizado
bValid	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.
lHScroll	Lógico, opcional. Se .T., habilita barra de rolagem horizontal.
lVScroll	Lógico, opcional. Se .T., habilita barra de rolagem vertical.

☑ **Métodos auxiliares:**

GoUp

- **Descrição:** Salta uma linha para cima.
- **Sintaxe: GoUp**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoDown

- **Descrição:** Salta uma linha para baixo.
- **Sintaxe: GoDown**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoTop

- **Descrição:** Salta para primeira linha.
- **Sintaxe: GoTop**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoBottom

- **Descrição:** Salta para ultima linha.
- **Sintaxe: GoBottom**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

RowCount

- **Descrição:** Retorna numero de linhas visiveis.
- **Sintaxe: RowCount( )**
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
user function TcBrowse_EX()
Local oOK   := LoadBitmap(GetResources(),'br_verde')
Local oNO   := LoadBitmap(GetResources(),'br_vermelho')
Local aList := {} // Vetor com elementos do Browse
Local nX
      // Cria Vetor para teste
for nX := 1 to 100
      aListAux := {.T., strzero(nX,10), 'Descrição do Produto '+;
      strzero(nX,3), 1000.22+nX}
      aadd(aList, aListAux)
next

DEFINE MSDIALOG oDlg FROM 0,0 TO 520,600 PIXEL TITLE 'Exemplo da TCBrowse'
      // Cria objeto de fonte que sera usado na Browse
      Define Font oFont Name 'Courier New' Size 0, -12
      // Cria Browse
      oList := TCBrowse():New( 01 , 01, 300, 200,,;
      '','Codigo','Descrição','Valor'},{20,50,50,50},;
      oDlg,,,,,{||},,oFont,,,,,.F.,,.T.,,.F.,,, )
      // Seta o vetor a ser utilizado
      oList:SetArray(aList)
      // Monta a linha a ser exibina no Browse
      oList:bLine := {||{ If(aList[oList:nAt,01],oOK,oNO),;
             aList[oList:nAt,02],;
             aList[oList:nAt,03],;
             Transform(aList[oList:nAT,04],'@E 99,999,999,999.99') } }
// Evento de DuploClick (troca o valor do primeiro elemento do Vetor)
              oList:bLDblClick := {|| aList[oList:nAt][1] :=;
              !aList[oList:nAt][1],oList:DrawSelect() }
            // Principais commandos
oBtn  := TButton():New( 210, 001,'GoUp()' , oDlg,{||oList:GoUp()},;
      40, 010,,,.F.,.T.,.F.,,.F.,,,.F. )
oBtn  := TButton():New( 220, 001,'GoDown()', oDlg,{||oList:GoDown()},;
      40, 010,,,.F.,.T.,.F.,,.F.,,,.F. )
oBtn  := TButton():New( 230, 001,'GoTop()' , oDlg,{||oList:GoTop()}, ;
      40, 010,,,.F.,.T.,.F.,,.F.,,,.F. )
oBtn  := TButton():New( 240, 001,'GoBottom()', oDlg,{||oList:GoBottom()},;
      40, 010,,,.F.,.T.,.F.,,.F.,,,.F. )
oBtn  := TButton():New( 210, 060, 'nAt (Linha selecionada)'   ,;
       oDlg,{|| Alert(oList:nAt)},;
       90,  010,,,.F.,.T.,.F.,,.F.,,,.F. )
oBtn  := TButton():New( 220, 060, 'nRowCount (Nr de linhas visiveis)',
oDlg,;{|| Alert(oList:nRowCount()) }, 90, 010,,,.F.,.T.,.F.,,.F.,,,.F. )

oBtn  := TButton():New( 230, 060, 'nLen (Numero total de linhas)', oDlg,;
{|| Alert(oList:nLen) }, 90, 010,,,.F.,.T.,.F.,,.F.,,,.F. )

oBtn  := TButton():New( 240, 060, 'lEditCell (Edita a celula)', oDlg,;
{|| lEditCell(@aList,oList,'@!',3) }, 90, 010,,,.F.,.T.,.F.,,.F.,,,.F. )
ACTIVATE MSDIALOG oDlg CENTERED
return
```

TCHECKBOX()

☑ **Descrição:** Classe de objetos visuais do tipo controle - CheckBox.

☑ **Propriedades:**

bLClicked	Bloco de código disparado ao clique do mouse no objeto
bSetGet	Bloco de código disparado na mudança de item selecionado, responsavel pela mudança de valor da variavel numerica que indica o item selecionado.
bWhen	Bloco de código recebe um lógico e de True permite alteração, se False não permite.
bValid	Bloco de código executado na saida do objeto.

☑ **Construtor:** New([nRow], [nCol], [cCaption], [bSetGet], [oDlg], [nWidth], [nHeight], [aHelpIds], [bLClicked], [oFont], [bValid], [nClrText], [nClrPane], [lDesign], [lPixel], [cMsg], [lUpdate], [bWhen])

☑ **Parâmetros:**

nRow	Numérico, opcional. Coordenada vertical
nCol	Numérico, opcional. Coordenada horizontal
cCaption	Caracter, Texto descritivo
bSetGet	Code-block, opcional, Responsável pela setagem de valor
oDlg	Objeto, opcional. Janela ou controle onde o divisor deverá ser criado.
nWidth	Numérico, opcional. Largura do objeto
nHeight	Numérico, opcional. Altura do objeto
aHelpIds	Não utilizado
bLClicked	Bloco de código, opcional. Executado ao clique do mouse.
oFont	Objeto, opcional. Fonte
bValid	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.
nClrText	Numérico, opcional. Cor do texto da janela.
nClrPane	Numérico, opcional. Cor de fundo da janela.
lDesign	Não utilizado
lPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
cMsg	Caracter, opcional. Mesamge ao posicionar o mouse sobre o objeto
lUpdate	Não utilizado
bWhen	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.

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
user function TCheckBox()
      DEFINE MSDIALOG oDlg FROM 0,0 TO 270,400 PIXEL TITLE 'Exemplo da
TCBrowse'
            lCheck1 := .T.
            oCheck1 := TCheckBox():New(01,01,'CheckBox 001',,oDlg,
100,210,,,,,,,,.T.,,,)
            oCheck2 := TCheckBox():New(11,01,'CheckBox 002',,oDlg,
100,210,,,,,,,,.T.,,,)
            oCheck3 := TCheckBox():New(21,01,'CheckBox 003',,oDlg,
100,210,,,,,,,,.T.,,,)

oCheck4 :=TCheckBox():New(31,01,'CheckBox 004',,oDlg,
100,210,,,,,,,,.T.,,,)
oCheck5 := TCheckBox():New(41,01,'CheckBox
005',,oDlg,100,210,,,,,,,,.T.,,,)


            // Seta Eventos do primeiro Check
            oCheck1:bSetGet   := {|| lCheck1 }
            oCheck1:bLClicked := {|| lCheck1:=!lCheck1 }
            oCheck1:bWhen           := {|| .T. }
            oCheck1:bValid          := {|| Alert('bValid') }
            // Principais comandos
            oBtn  := TButton():New( 060, 001, 'Retorna estado do CheckBox
001',;
                  oDlg,{|| Alert(lCheck1) }, 120,
010,,,.F.,.T.,.F.,,.F.,,,.F. )
        ACTIVATE MSDIALOG oDlg CENTERED
return
```

TCOLORTRIANGLE()

☑ **Descrição:** Classe de objetos visuais do tipo Paleta de Cores.

☑ **Propriedades:**

-	Herdadas das classes superiores.

☑ **Construtor:** New([anRow], [anCol], [aoWnd], [anWidth], [anHeight])

☑ **Parâmetros:**

anRow	Numérico, opcional. Coordenada vertical em pixels ou carateres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
aoWnd	Objeto, opcional. Janela ou controle onde a paleta deverá ser criada.
anWidth	Numérico, opcional. Largura da paleta em pixels.
anHeight	Numérico, opcional. Altura da paleta em pixels.

☑ **Métodos auxiliares:**

Create

- **Descrição:** Método construtor opcional da classe.
- **Sintaxe:** Create(aoWnd)
- **Parâmetros:**

aoWnd	Objeto, opcional. Janela ou controle onde a paleta deverá ser criada.

- **Retorno:**

Objeto	Objeto do tipo TColorTriangule com os atributos definidos com conteúdo padrão.

RetColor

- **Descrição:** Retorna o RGB da cor selecionada
- **Sintaxe: RetColor** ( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Numérico	Valor que representa do RGB da cor.

SetColorIni

- **Descrição:** Define a cor inicial selecionada para o controle.
- **Sintaxe: SetColorIni** (nColor)
- **Parâmetros:**

nColor	Valor da cor inicial no padrão RGB.

- **Retorno:**

Nenhum	-

SetColor

- **Descrição:** Define a cor corrente.
- **Sintaxe: SetColor** (nColor)
- **Parâmetros:**

nColor	Valor da cor inicial no padrão RGB.

- **Retorno:**

Nenhum	-

SetSizeTriangule

- **Descrição:** Define o tamanho do triângulo de configuração das cores.
- **Sintaxe: SetSizeTriangule** (nWidh, nHeight)
- **Parâmetros:**

nWidth	Numérico. Largura
nHeight	Numérico.Altura

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
#include "protheus.ch"


User Function tColor()
Local oDlg, oColorT
DEFINE MSDIALOG oDlg FROM 0,0 TO 500,600 PIXEL TITLE "Cores"
// Usando o método create
oColorT := tColorTriangle():Create( oDlg  )
oColorT:SetColorIni( nColorIni )
ACTIVATE MSDIALOG oDlg CENTERED
Return Nil
```

TCOMBOBOX()

☑ **Descrição:** 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.

☑ **Propriedades:**

aItems	Array. Lista de itens, caracteres, a serem exibidos. Pode ter os seguintes formatos: a) Seqüencial, exemplo: {"item1","item2",...,"itemN"} ou b) Indexada, exemplo: {"a=item1","b=item2", ..., "n=itemN"}.
nAt	Numérico. Posição do item selecionado.

☑ **Construtor:** New([anRow], [anCol], [abSetGet], [anItems], [anWidth], [anHeight], [aoWnd], [nPar8], [abChange], [abValid], [anClrText], [anClrBack], [alPixel], [aoFont], [cPar15], [lPar16], [abWhen], [lPar18], [aPar19], [bPar20], [cPar21], [acReadVar])

☑ **Parâmetros:**

Parâmetro	
anRow	Numérico, opcional. Coordenada vertical em pixels ou caracteres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
abSetGet	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. Se a lista for seqüencial, o controle atualizará <var> com o conteúdo do item selecionado, se a lista for indexada, <var> será atualizada com o valor do índice do item selecionado.
anItems	Array, opcional. Lista de items, caracteres, a serem exibidos. Pode ter os seguintes formatos: a) Seqüencial, exemplo: {"item1","item2",...,"itemN"} ou b) Indexada, exemplo: {"a=item1","b=item2", ..., "n=itemN"}.
anWidth	Numérico, opcional. Largura do controle em pixels.

anHeight	Numérico, opcional. Altura do controle em pixels.
aoWnd	Objeto, opcional. Janela ou controle onde o controle será criado.
nPar8	Reservado.
abChange	Bloco de código, opcional. Executado quando o controle modifica o item selecionado.
abValid	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.
anClrBack	Numérico, opcional. Cor de fundo do controle.
anClrText	Numérico, opcional. Cor do texto do controle.
alPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
aoFont	Objeto, opcional. Objeto tipo tFont utilizado para definir as características da fonte utilizada para exibir o conteúdo do controle.
cPar15	Reservado.
lPar16	Reservado.
abWhen	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.
lPar18	Reservado.
aPar19	Reservado.
bPar20	Reservado.
cPar21	Reservado.
acReadVar	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( ).

☑ **Métodos auxiliares:**

Select

- **Descrição:** Muda o item selecionado no combobox.
- **Sintaxe:** Select([anItem])
- **Parâmetros:**

anItem	Numérico, opcional. Posição do item a ser selecionado.

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function TesteGet()
Local oDlg, oButton, oCombo, cCombo
aItems:= {'item1','item2','item3'}
cCombo:= aItems[2]
DEFINE MSDIALOG oDlg FROM 0,0 TO 300,300 PIXEL TITLE 'Meu Combo'
 oCombo:= tComboBox():New(10,10,{|u|if(PCount()>0,cCombo:=u,cCombo)},;
 aItems,100,20,oDlg,,{||MsgStop('Mudou item')},,,,.T.,,,,,,,,,,'cCombo')

 // Botão para fechar a janela
 oButton:=tButton():New(30,10,'fechar',oDlg,{||oDlg:End()},100,20,,,,.T.)
ACTIVATE MSDIALOG oDlg CENTERED
MsgStop( 'O valor é '+cCombo )
Return NIL
```

TFOLDER()

☑ **Descrição:** Classe de objetos visuais do tipo controle - Folder.

☑ **Propriedades:**

aPrompts	Array, Titulo dos folders
aDialogs	Array, Nome dos diálogos
nOption	Numérico, Folder selecionado
bSetOption	Bloco de código, Executado na seleção do folder

☑ **Construtor:** New([anTop], [anLeft], [aPrompts], [aDialogs], [aoWnd], [anOption], [anClrFore], [anClrBack], [alPixel], [alDesign], [anWidth], [anHeight], [acMsg], [alAllWidth])

☑ **Parâmetros:**

anTop	Numérico, opcional. Coordenada vertical em pixels ou carateres.
anLeft	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
aPrompts	Array, Titulo dos folders
aDialogs	Array, Nome dos diálogos
aoWnd	Objeto, opcional. Janela ou controle onde a botão deverá ser criado
anOption	Numérico, opcional. Folder selecionado
anClrFore	Numérico, opcional. Cor de frente
anClrBack	Numérico, opcional. Cor de fundo
alPixel	Lógico, opcional. Utiliza coordenadas em pixel
alDesign	Lógico, opcional. NÃO USADO
anWidth	Numérico, opcional. Largura em pixels.
anHeight	Numérico, opcional. Altura em pixels.
acMsg	Caractere, Mensagem de Hint
alAllWidth	Lógico, opcional. NÃO USADO

☑ **Métodos auxiliares:**

SetOption

- **Descrição:** Seleciona folder desejado
- **Sintaxe:** SetOption(nOption)
- **Parâmetros:**

nOption	Numérico. Folder a ser selecionado

- **Retorno:**

Nenhum	-

AddItem

- **Descrição:** Insere um folder
- **Sintaxe: AddItem** (cItem, lVisible)
- **Parâmetros:**

cItem	Caractere. Título do Folder
lVisible	Lógico. Visível

- **Retorno:**

Nenhum	-

aEnable

- **Descrição:** Habilita/Desabilita um folder
- **Sintaxe:** aEnable(nItem, lEnable)
- **Parâmetros:**

nItem	Numérico. Folder para habilitar/desabilitar
lEnable	Lógico. Habilita/Desabilita

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
Local oFolder
Local aFolder := { 'Folder 1', 'Folder 2', 'Folder 3' }

oFolder := TFolder():New( 0, 0, aFolder, aFolder, oDlg,,,, .T., , 200,;
           200 )
```

TGET()

☑ **Descrição:** 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.

☑ **Propriedades:**

lPassword	Lógico. Se .T. o controle se comporta como entrada de dados de senha, exibindo asteriscos '*' para esconder o conteúdo digitado.
Picture	Caractere. Máscara de formatação do conteúdo a ser exibido.

☑ **Construtor:** New([anRow], [anCol], [abSetGet], [aoWnd], [anWidth], [anHeight], [acPict], [abValid], [anClrFore], [anClrBack], [aoFont], [lPar12], [oPar13], [alPixel], [cPar15], [lPar16], [abWhen], [lPar18], [lPar19], [abChange], [alReadOnly], [alPassword], [cPar23], [acReadVar], [cPar25], [lPar26], [nPar27], [lPar28])

☑ **Parâmetros:**

anRow	Numérico, opcional. Coordenada vertical em pixels ou caracteres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
abSetGet	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.
aoWnd	Objeto, opcional. Janela ou controle onde o controle será criado.
anWidth	Numérico, opcional. Largura do controle em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
acPict	Caractere, opcional. Máscara de formatação do conteúdo a ser exibido.
abValid	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.
anClrFore	Numérico, opcional. Cor de fundo do controle.
anClrBack	Numérico, opcional. Cor do texto do controle.
aoFont	Objeto, opcional. Objeto tipo tFont utilizado para definir as características da fonte utilizada para exibir o conteúdo do

	controle.
lPar12	Reservado.
oPar13	Reservado.
alPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
cPar15	Reservado.
lPar16	Reservado.
abWhen	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.
lPar18	Reservado.
lPar19	Reservado.
abChange	Bloco de código, opcional. Executado quando o controle modifica o valor da variável associada.
alReadOnly	Lógico, opcional. Se .T. o controle não poderá ser editado.
alPassword	Lógico, opcional. Se .T. o controle exibirá asteriscos "*" no lugar dos caracteres exibidos pelo controle para simular entrada de senha.
cPar23	Reservado.
acReadVar	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.
lPar26	Reservado.
nPar27	Reservado.
lPar28	Reservado.

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function TesteGet()
Local oDlg, oButton, oCombo, cCombo, nGet1:=0
DEFINE MSDIALOG oDlg FROM 0,0 TO 300,300 PIXEL TITLE 'Meu Get'
 oGet1:= TGet():New(10,10,{|u| if(PCount()>0,nGet1:=u,nGet1)}, oDlg,;
 100,20,'@E 999,999.99',;
 {|o|nGet1>1000.00},,,,,,.T.,,,,,,,,,,'nGet1')

 // Botão para fechar a janela
 oButton:=tButton():New(30,10,'fechar',oDlg,{||oDlg:End()},100,20,,,,.T.)
ACTIVATE MSDIALOG oDlg CENTERED
MsgStop( 'O valor é '+Transform(nGet1,'@E 999,999.00') )

Return NIL
```

TGROUP()

☑ **Descrição:** 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.

☑ **Propriedades:**

-	Herdadas das classes superiores.

☑ **Construtor:** New([anTop], [anLeft], [anBottom], [anRight], [acCaption], [aoWnd], [anClrText], [anClrPane], [alPixel], [lPar10])

☑ **Parâmetros:**

anTop	Numérico, opcional. Coordenada vertical superior em pixels ou caracteres.
anLeft	Numérico, opcional. Coordenada horizontal esquerda em pixels ou caracteres.
anBottom	Numérico, opcional. Coordenada vertical inferior em pixels ou caracteres.
anRight	Numérico, opcional. Coordenada horizontal direita em pixels ou caracteres.
acCaption	Caractere, opcional. Título do grupo.
aoWnd	Objeto, opcional. Janela ou controle onde o controle será criado.
anClrText	Numérico, opcional. Cor do texto.
anClrPane	Numérico, opcional. Cor do fundo.
alPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
lPar10	Reservado.

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg, oButton, oCombo, cCombo, cGet1:='Teste'
DEFINE MSDIALOG oDlg FROM 0,0 TO 300,300 PIXEL TITLE 'Meu test'
 oGroup:= tGroup():New(10,10,130,130,'grupo de gets',oDlg,,,.T.)
 @ 18,16 MSGET oGet1 VAR cGet1 SIZE 100,10 OF oGroup PIXEL
 @ 38,16 MSGET oGet2 VAR cGet1 SIZE 100,10 OF oGroup PIXEL
ACTIVATE MSDIALOG oDlg CENTERED
```

Anotações

THBUTTON()

☑ **Descrição:** Classe de objetos visuais do tipo botão com hiperlink.

☑ **Propriedades:**

bAction	Bloco de código. Executado quando o botão é pressionado.

☑ **Construtor:** New([anRow], [anCol], [acCaption], [aoWnd], [abAction], [anWidth], [anHeight], [aoFont], [abWhen])

☑ **Parâmetros:**

Parâmetro	Tipo / Descrição
anRow	Numérico, opcional. Coordenada vertical em pixels ou carateres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
acCaption	Caractere, opcional. Titulo do botão.
aoWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
abAction	Bloco de código, opcional. Bloco que deverá ser acionado quando o botão for pressionado.
anWidth	Numérico, opcional. Largura do botão em pixels.
anHeight	Numérico, opcional. Altura do botão em pixels.
aoFont	Objeto, opcional. Objeto tipo tFont com propriedades da fonte utilizada para o título do botão.
abWhen	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.

☑ **Métodos auxiliares:**

Create

- **Descrição:** Método construtor opcional da classe.
- **Sintaxe:** Create(aoWnd)
- **Parâmetros:**

aoWnd	Objeto, opcional. Janela ou controle onde a paleta deverá ser criada.

- **Retorno:**

Objeto	Objeto do tipo THButton com os atributos definidos com conteúdo padrão.

☑ **Aparência:**

Exemplo:

```
#include “protheus.ch”
#include “hbutton.ch”
User Function MyhBtn()
Local oDlg, ohBtn
DEFINE MSDIALOG oDlg FROM 0,0 TO 500,600 PIXEL TITLE “Meu hButton”
// Usando o método create
ohBtn:= tHButton():Create( oDlg )
ohBtn:nTop      := 100
ohBtn:nLeft     := 10
ohBtn:nWidth    := 100
ohBtn:nHeight   := 30
ohBtn:cCaption  := 'hButton'
ohBtn:blClicked := { || MsgStop( 'Cliquei' ) }
// Usando o command
@ 200,100 HBUTTON ohBtn PROMPT 'Exemplo hButton' SIZE 100,30 ACTION
MsgStop('Cliquei') OF oDlg MESSAGE 'hButton'
ACTIVATE MSDIALOG oDlg CENTERED
Return Nil
```

TIBROWSER()

☑ **Descrição:** Classe de objetos visuais do tipo Página de Internet, sendo necessário incluir a clausula BrowserEnabled=1 no Config do Remote.INI

☑ **Propriedades:**

-	Herdadas da classe superior

☑ **Construtor:** New([nRow], [nCol], [nWidth], [nHeight], [cPager], [oWnd])

☑ **Parâmetros:**

nRow	Numérico, opcional. Coordenada vertical em pixels ou carateres.
nCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres
nWidth	Numérico, opcional. Largura em pixels
nHeigth	Numérico, opcional. Altura em pixels.
cPage	Caracter. Enredeço URL da página de Internet
oWnd	Objeto, opcional. Janela ou controle onde a botão deverá ser criado

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 304,504 PIXEL TITLE 'Exemplo'

      // Habilitar no Remote(Config) -> BrowserEnabled=1
      oTIBrowser:=
TIBrowser():New(0,0,306,134,'http://www.microsiga.com.br',oDlg )

      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

Anotações

TLISTBOX()

☑ **Descrição:** 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.

☑ **Propriedades:**

nAt	Numérico. Posição do item selecionado.
aItems	Array de items caracteres. Lista do itens selecionáveis.

☑ **Construtor:** New([anRow], [anCol], [abSetGet], [aaItems], [anWidth], [anHeigth], [abChange], [aoWnd], [abValid], [anClrFore], [anClrBack], [alPixel], [lPar13], [abLDBLClick], [aoFont], [cPar16], [lPar17], [abWhen], [aPar19], [bPar20], [lPar21], [lPar22], [abRightClick])

☑ **Parâmetros:**

anRow	Numérico, opcional. Coordenada vertical em pixels ou caracteres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
abSetGet	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 ou numérica.
aaItems	Array de items caracteres, opcional. Lista de items selecionáveis.
anWidth	Numérico, opcional. Largura do controle em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
abChange	Bloco de código, opcional. Executado quando o item selecionado é alterado.
aoWnd	Objeto, opcional. Janela ou controle onde o controle será criado.
abValid	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.
anClrFore	Numérico, opcional. Cor de fundo do controle.
anClrBack	Numérico, opcional. Cor do texto do controle.
alPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
lPar13	Reservado.
abLDBLClick	Bloco de código, opcional. Executado quando acionado duplo click do botão esquerdo do mouse sobre o controle.
aoFont	Objeto, opcional. Objeto tipo tFont utilizado para definir as características da fonte utilizada para exibir o conteúdo do controle.
cPar16	Reservado.
lPar17	Reservado.
abWhen	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.

aPar19	Reservado.
bPar20	Reservado.
lPar21	Reservado.
lPar22	Reservado.
abRightClick	Bloco de código, opcional. Executado quando acionado click do botão direito do mouse sobre o controle.

☑ **Métodos auxiliares:**

Select

- **Descrição:** Força a seleção de um item.
- **Sintaxe:** Select([anItem])
- **Parâmetros:**

nItem	Numérico, opcional. Posição do item a ser selecionado.

- **Retorno:**

Nenhum	-

Add

- **Descrição:** Adiciona novo item.
- **Sintaxe:** Add(cText, nPos)
- **Parâmetros:**

cText	Caractere, obrigatório. Texto do item.
nPos	Numérico, obrigatório. Se 0 ou maior que o número de itens, insere o item no final da lista. Se valor entre 1 e número de itens, insere o item na posição informada, empurrando o item anterior para baixo.

- **Retorno:**

Nenhum	-

Modify

- **Descrição:** Modifica o texto de um item.
- **Sintaxe:** Modify(cText, nPos)
- **Parâmetros:**

cText	Caractere, obrigatório. Novo texto do item.
nPos	Numérico, obrigatório. Posição a ser modificada deve ser maior que 0 e menor ou igual que o número de itens.

- **Retorno:**

Nenhum	-

Del

- **Descrição:** Remove um item.
- **Sintaxe:** Del(nPos)
- **Parâmetros:**

nPos	Numérico, obrigatório. Posição a ser excluida, deve ser maior que 0 e menor ou igual que o número de itens.

- **Retorno:**

Nenhum	-

Len

- **Descrição:** Retorna o número de itens.
- **Sintaxe:** Len( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Numérico	Número de itens.

Reset

- **Descrição:** Apaga todos os itens.
- **Sintaxe:** Reset( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg, oList, nList:= 1
Local aItems:={}
Aadd(aItems,'Item 1')
Aadd(aItems,'Item 2')
Aadd(aItems,'Item 3')
Aadd(aItems,'Item 4')
DEFINE MSDIALOG oDlg FROM 0,0 TO 400,400 PIXEL TITLE 'Teste'
 oList:= tListBox():New(10,10,{|u|if(Pcount()>0,nList:=u,nList)};
      ,aItems,100,100,,oDlg,,,,.T.)
ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

TMENU()

☑ **Descrição:** Classe de objetos visuais do tipo controle - Menu.

☑ **Propriedades:**

-	Herdadas das classes superiores.

☑ **Construtor:** New ([nTop], [nLeft], [nHeight], [nWidth], [lPopUp], [cBmpName], [oWnd], [nClrNoSelect], [nClrSelect], [cArrowUpNoSel], [cArrowUpSel], [cArrowDownNoSel], [cArrowDownSel])

☑ **Parâmetros:**

nTop	Numérico, opcional. Coordenada vertical em pixels.
nLeft	Numérico, opcional. Coordenada horizontal em pixels.
nHeight	Numérico, opcional. Altura do controle em pixels.
nWidth	Numérico, opcional. Largura do controle em pixels.
lPopUp	Lógico. Define se o objeto será um PoPup
cBmpName	Caracter opcional. Figura do Menu
oWnd	Objeto, opcional. Janela ou controle onde a botão deverá ser criado
nClrNoSelect	Numerico opcional. Cor quando não selecionado
nClrSelect	Numerico opcional. Cor quando selecionado
cArrowUpNoSel	Caracter opcional, Define a figura da seta para cima quando não selecionado o item.
cArrowUpSel	Caracter opcional, Define a figura da seta para cima quando selecionado o item.
cArrowDownNoSel	Caracter opcional, Define a figura da seta para baixo quando não selecionado o item.
cArrowDownSel	Caracter opcional, Define a figura da seta para baixo quando selecionado o item.

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 304,504 PIXEL TITLE 'Exemplo'
      oMenu := TMenu():New( 0,0,0,0,.F.,'',oDlg,CLR_WHITE,CLR_BLACK)
      // Adiciona Item ao Menu Principal
      oMenuItem1 := TMenuItem():New2( oMenu:Owner(),'Item 001','Item
001',,,)
      oMenu:Add( oMenuItem1 )

      // Adiciona sub-Itens
      oMenuItem2 := TMenuItem():New2( oMenu:Owner(),'Sub-Item
001',,,{||Alert('TMenuItem')})
      oMenuItem3 := TMenuItem():New2( oMenu:Owner(),'Sub-Item
002',,,{||Alert('TMenuItem')})
      oMenuItem1:Add( oMenuItem2 )
      oMenuItem1:Add( oMenuItem3 )

      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

TMENUBAR()

☑ **Descrição:** Classe de objetos visuais do tipo controle - Barra de Menu.

☑ **Propriedades:**

-	Herdadas das classes superiores.

☑ **Construtor:** New([oWnd])

☑ **Parâmetros:**

oWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 304,504 PIXEL TITLE 'Exemplo'
      // Monta um Menu Suspenso
      TMenuBar := TMenuBar():New(oDlg)
      TMenuBar:nClrPane := RGB(183,208,248) // Cor de fundo do Menu
      oTMenu1 := TMenu():New(0,0,0,0,.T.,,oDlg)
      oTMenu2 := TMenu():New(0,0,0,0,.T.,,oDlg)
      TMenuBar:AddItem('Arquivo'  , oTMenu1, .T.)
      TMenuBar:AddItem('Relatorio', oTMenu2, .T.)

      // Cria Itens do Menu
      oTMenuItem := TMenuItem():New(oDlg,'TMenuItem 01',,,,;
                     {||Alert('TMenuItem 01')},,'AVGLBPAR1',,,,,,,.T.)
      oTMenu1:Add(oTMenuItem)
      oTMenu2:Add(oTMenuItem)
      oTMenuItem := TMenuItem():New(oDlg,'TMenuItem 02',,,,;
                    {||Alert('TMenuItem 02')},,,,,,,,,.T.)
      oTMenu1:Add(oTMenuItem)
      oTMenu2:Add(oTMenuItem)

      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

TMETER()

☑ **Descrição:** 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.

☑ **Propriedades:**

nTotal	Numérico. Número total de passos até o preenchimento da régua de processo.
lPercentage	Lógico. Se .T. considera o passo de movimentação em porcentagem.
nClrBar	Numérico. Cor da barra de andamento.

☑ **Construtor:** New([anRow], [anCol], [abSetGet], [anTotal], [aoWnd], [anWidth], [anHeight], [lPar8], [alPixel], [oPar10], [cPar11], [alNoPerc], [anClrPane], [nPar14], [anClrBar], [nPar16], [lPar17])

☑ **Parâmetros:**

anRow	Numérico, opcional. Coordenada vertical em pixels ou caracteres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
abSetGet	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 numérico.
anTotal	Numérico, opcional. Numero total de passos até o preenchimento da régua de processo.

aoWnd	Objeto, opcional. Janela ou controle onde o controle sera criado.
anWidth	Numérico, opcional. Largura do controle em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
lPar8	Reservado.
alPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
oPar10	Reservado.
cPar11	Reservado.
alNoPerc	Lógico, opcional. Se .T. (padrão) não considera os passos de atualização em porcentagem.
anClrPane	Numérico, opcional. Cor de fundo do controle.
nPar14	Reservado.
anClrBar	Numérico, opcional. Cor da barra de andamento.
nPar16	Reservado.
lPar17	Reservado.

☑ **Métodos auxiliares:**

Set

- **Descrição:** Atualiza a posição da régua de processamento.
- **Sintaxe:** Set([nVal])
- **Parâmetros:**

nVal	Numérico, opcional. Novo valor da posição da régua de processamento.

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
STATIC lRunning:=.F., lStop:=.F.
User Function Teste()
Local oDlg, oMeter, nMeter:=0, oBtn1, oBtn2
 DEFINE MSDIALOG oDlg FROM 0,0 TO 400,400 TITLE 'Teste' Pixel
  oMeter:= tMeter():New(10,10,{|u|if(Pcount()>0,nMeter:=u,nMeter)};
    ,100,oDlg,100,16,,.T.) // cria a régua
  // botão para ativar andamento da régua
  @ 30,10 BUTTON oBtn1 PROMPT 'Run ' OF oDlg PIXEL ACTION RunMeter(oMeter)
  @ 50,10 BUTTON oBtn2 PROMPT 'Stop' OF oDlg PIXEL ACTION lStop:=.T.
 ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
STATIC Function RunMeter(oMeter)
 If lRunning
  Return
 Endif
 lRunning:= .T.

 oMeter:Set(0)
 // inicia a régua
 While .T. .and. !lStop
  Sleep(1000) // pára 1 segundo
  ProcessMessages() // atualiza a pintura da janela, processa mensagens do
windows
  nCurrent:= Eval(oMeter:bSetGet) // pega valor corrente da régua
  nCurrent+=10 // atualiza régua

  oMeter:Set(nCurrent)
  if nCurrent==oMeter:nTotal
   Return
  endif
 Enddo

 lRunning:= .F.
 lStop:= .F.
Return
```

TMSGRAPHIC()

☑ **Descrição:** Classe de objetos visuais do tipo controle - Gráfico.

☑ **Propriedades:**

l3D	Lógico, opcional Gráfico em 3D
lAxisVisib	Lógico, opcional Mostra eixos do gráfico

☑ **Construtor:** New([anRow], [anCol], [aoWnd], [aoFont], [anClrText], [anClrBack], [anWidth], [anHeight])

☑ **Parâmetros:**

anRow	Numérico, opcional. Coordenada vertical em pixels.
anCol	Numérico, opcional. Coordenada horizontal em pixels.
aoWnd	Objeto, opcional. Janela ou controle onde a imagem deverá ser criada.
aoFont	Objeto, opcional. Fonte utilizada no gráfico.
anClrText	Caractere, Nome do resource.
anClrBack	Caractere, Nome do arquivo.
anWidth	Lógico, opcional. Exibe borda
anHeight	Objeto, opcional. Janela ou controle onde a imagem deverá ser criada.

☑ **Métodos auxiliares:**

CreateSerie

- **Descrição:** Método para criação de uma serie para o gráfico.
- **Sintaxe:** CreateSerie([nSerieType], [cLegend], [nDecimals], [lShowValues])
- **Parâmetros:**

nSerieType	Numérico. Indica o tipo do gráfico GRP_LINE 1 GRP_AREA 2 GRP_POINT 3 GRP_BAR 4 GRP_PIE 10
cLegend	Caractere, opcional. Legenda da série.
nDecimals	Numérico, opcional. Numero de casas decimais dos valores.
lShowValues	Lógico, opcional. Mostra valores

- **Retorno:**

Numérico	Numero da série criada.

Add

- **Descrição:** Método para adicionar um item ao gráfico.
- **Sintaxe:** Add(nSerie, nVal, cLegend, nColor)
- **Parâmetros:**

nSerie	Numérico. Serie a ser inserido o item.
nVal	Numérico. Valor do item.
cLegend	Caractere. Legenda do item.
nColor	Numérico, Cor do item.

- **Retorno:**

Nenhum	-

SetGradiente

- **Descrição:** Método para setar o fundo do gráfico com uma cor gradiente em determinada direção.
- **Sintaxe:** SetGradient(nDirection, StartColor, EndColor)
- **Parâmetros:**

nDirection	Numérico. Indica direção do gradiente. GDTOPBOTTOM 1 GDBOTTOMTOP 2 GDLEFTRIGHT 3 GDRIGHTLEFT 4
StartColor	Numérico. Cor inicial.
EndColor	Numérico. Cor final.

- **Retorno:**

Nenhum	-

SetTitle

- **Descrição:** Define o titulo do gráfico.
- **Sintaxe:** SetTitle(cTitle, cTitle2, Color, Aligment, lFoot)
- **Parâmetros:**

cTitle	Caractere. Primeiro titulo.
cTitle2	Caractere. Segundo titulo.
Color	Numérico. Cor do titulo.
Aligment	Numérico. Alinhamento do titulo. A_LEFTJUST 1 A_RIGHTJUS 2 A_CENTER 3
lFoot	Lógico. Indica titulo no rodapé do gráfico.

- **Retorno:**

Nenhum	-

SetLegenProp

- **Descrição:** Método para setar propriedades da legenda.
- **Sintaxe:** SetLegenProp(Align, Color, Style, Visible)
- **Parâmetros:**

Align	Numerico. Alinhamento da legenda. GRP_SCRTOP 1 GRP_SCRLEFT 2 GRP_SCRBOTTOM 3 GRP_SCRRIGHT 4
Color	Numérico. Cor da legenda.
Style	Numérico. Estilo da legenda. GRP_AUTO 1 GRP_SERIES 2 GRP_VALUES 3 GRPLASTVAL 4
Visible	Lógico. Indica de o titulo será visível.

- **Retorno:**

Nenhum	-

SetMargins

- **Descrição:** Método para setar as margens superior, inferior e laterais do gráfico.
- **Sintaxe:** SetMargins(nTop, nLeft, nBottom, nRight)
- **Parâmetros:**

nTop	Numérico. Posição em relação ao topo do gráfico.
nLeft	Numérico. Posição em relação a esquerda.
nBottom	Numérico. Posição em relação ao rodapé.
nRight	Numérico. Posição em relação a direita.

- **Retorno:**

Nenhum	-

SetRangeY

- **Descrição: Define a escala dos valores do eixo Y**
- **Sintaxe:** SetRangeY (min, max, delta).
- **Parâmetros:**

Min	valor inicial
Max	valor final
delta	intervalo entre os valores [opcional, calculado automaticamente]

- **Retorno:**

Nenhum	-

SaveToBMP

- **Descrição:** Método para salvar o gráfico atual em um bitmap no server(**Obsoleto, ver SaveToImage**).
- **Sintaxe:** SaveToBMP(BmpName, PathToWrite)
- **Parâmetros:**

BmpName	Caractere, Nome do da figura a ser salva.
PathToWrite	Caractere. Path no Server onde será salva a figura.

- **Retorno:**

Lógico	Indica se a imagem foi salva corretamente.

SaveToImage

- **Descrição:** Método para salvar o gráfico atual em um formato de pré-determinado no server.
- **Sintaxe:** SaveToImage(BmpName, PathToWrite, TypeImage)
- **Parâmetros:**

BmpName	Caractere, Nome do da figura a ser salva.
PathToWrite	Caractere. Path no Server onde será salva a figura.
TypeImage	Caractere. Tipo da Figura (Default 'JPEG') Tipos suportados: JPEG, PNG, BMP

- **Retorno:**

Lógico	Indica se a imagem foi salva corretamente.

DelSerie

- **Descrição:** Método para deletar uma série do grafico.
- **Sintaxe:** DelSerie (nSerie)
- **Parâmetros:**

nSerie	Numérico. Serie a ser deletada.

- **Retorno:**

Lógico	Indica se a séria foi removida do gráfico

ZoomIn

- **Descrição:** Método para efetuar zoom interno (+).
- **Sintaxe:** ZoomIn()
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nenhum	-

ZoomOut

- **Descrição:** Método para efetuar zoom externo (-).
- **Sintaxe:** ZoomOut()
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Anotações

Exemplo:

```
#include 'MSGRAPHI.CH'
User Function tMsGraphicTst()

      DEFINE MSDIALOG oDlg TITLE '' FROM 0,0 TO 250,330 PIXEL

      @ 001, 001 MSGRAPHIC oGraphic SIZE 120,120 OF oDlg

      oGraphic:SetTitle('Titulo do Grafico', DTOC(Date()), CLR_BLACK,
A_LEFTJUST, GRP_TITLE )
      oGraphic:SetMargins(2,6,6,6)
      oGraphic:SetGradient(GDBOTTOMTOP, CLR_HGRAY, CLR_WHITE)
      oGraphic:SetLegenProp(GRP_SCRRIGHT, CLR_HGRAY, GRP_AUTO,.T.)

      nSerie      :=    oGraphic:CreateSerie( 10 )
      oGraphic:l3D := .T. // Grafico em 3D
      oGraphic:lAxisVisib     := .T.      // Mostra os eixos

      // Itens do Grafico
      oGraphic:Add(nSerie, 200, 'Item01', CLR_BLUE )
      oGraphic:Add(nSerie, 180, 'Item02', CLR_RED  )
      oGraphic:Add(nSerie, 210, 'Item03', CLR_BLUE )
      oGraphic:Add(nSerie, 110, 'Item04', CLR_BLUE )
      oGraphic:Add(nSerie, 100, 'Item05', CLR_BLUE )
      oGraphic:Add(nSerie, 080, 'Item06', CLR_BLUE )
      oGraphic:Add(nSerie, 110, 'Item07', CLR_BLUE )

      @ 001, 124 BUTTON 'Salva Grafico' SIZE 40,14 OF oDlg PIXEL ACTION
oGraphic:SaveToBMP('Exemplo.bmp','\web\')
      @ 020, 124 BUTTON 'Zoom IN'       SIZE 40,14 OF oDlg PIXEL ACTION
oGraphic:ZoomIn()
      @ 040, 124 BUTTON 'Zoom OUT'      SIZE 40,14 OF oDlg PIXEL ACTION
oGraphic:ZoomOut()
      @ 060, 124 BUTTON 'Deleta Serie'  SIZE 40,14 OF oDlg PIXEL ACTION
oGraphic:DelSerie(nSerie)

      ACTIVATE MSDIALOG oDlg CENTERED

Return
```

Anotações

TMSGBAR()

☑ **Descrição:** Classe de objetos visuais do tipo controle - Rodapé.

☑ **Propriedades:**

-	Herdadas das classes superiores.

☑ **Construtor:** New ([oWnd], [cPrompt,], [lCentered], [lClock], [lDate], [lKbd], [nClrFore], [nClrBack], [oFont], [lInset], [imageName])

☑ **Parâmetros:**

oWnd	Objeto, opcional. Janela ou controle onde a botão deverá ser criado
cPrompt	Caracter. Descrição na Barra
lCentered	Logico. Define centralização do texto
lClock	Nao utilizado
lDate	Nao utilizado
lKbd	Nao utilizado
nClrFore	Numerico, opcional. Define cor da fonte da barra
nClrBack	Numerico, opcional. Define cor do fundo da barra
oFont	Objeto, opcional. Objeto tipo tFont com propriedades da fonte utilizada para o título do botão.
lInset	Nao utilizado
imageName	Caracter, opcional. Insere figura lateral esquerda

☑ **Métodos auxiliares:**

AddItem

- **Descrição:** Método insere um subitem no rodapé
- **Sintaxe:** AddItem(oTMsgItem)
- **Parâmetros:**

oTMsgItem	Objeto do tipo TMsgItem que será inserido como subitem do rodapé.

- **Retorno:**

Nenhum	-

SetMsg

- **Descrição:** Define a descrição da barra de rodapé
- **Sintaxe:** SetMsg(cTexto)
- **Parâmetros:**

cTexto	Texto a ser inserido na barra

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 304,504 PIXEL TITLE 'Exemplo'
      oTMsgBar := TMsgBar():New(oDlg, 'MP10 | Totvs/Software',
.F.,.F.,.F.,.F., RGB(116,116,116),,,.F.)
      oTMsgItem1 := TMsgItem():New( oTMsgBar,'oTMsgItem1', 204,,,, .T.,
{||})
oTMsgItem2 := TMsgItem():New( oTMsgBar,'oTMsgItem2', 040,,,, .T.,
{||Alert('Item da Barra Acionado')} )

      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

TMSGITEM()

☑ **Descrição:** Classe de objetos visuais do tipo controle – utilizado para adicionar mensagens em outros objetos, como barras e menus.

☑ **Propriedades:**

-	Herdadas das classes superiores

☑ **Construtor: New(<oMsgBar>, <cMsg>, <nSize>, <oFont>, <nClrFore>, <nClrBack>, lEnable , [<{bAction}>], cImagem)**

☑ **Parâmetros:**

oMsgBar	Objeto do tipo barra ou menu no qual será adicionado o item de mensagem.
cMsg	Mensagem que será exibida pelo objeto
nSize	Tamanho da fonte do texto
oFont	Fonte do texto
nClrFore	Cor da fonte do texto
nClrBack	Cor do fundo do objeto
lEnable	Indica se o objeto está habilitado.
bAction	Bloco de código executado quando o objeto é ativado
cImagem	Imagem a ser vinculada no objeto.

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 304,504 PIXEL TITLE 'Exemplo'
      oTMsgBar := TMsgBar():New(oDlg, 'MP10 | Totvs/Software',
.F.,.F.,.F.,.F., RGB(116,116,116),,,.F.)
      oTMsgItem1 := TMsgItem():New( oTMsgBar,'oTMsgItem1', 204,,,, .T.,
{||})
oTMsgItem2 := TMsgItem():New( oTMsgBar,'oTMsgItem2', 040,,,, .T.,
{||Alert('Item da Barra Acionado')} )

      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL

.
```

TMULTIBTN()

☑ **Descrição:** Classe de objetos visuais do tipo controle - Múltiplos botões.

☑ **Propriedades:**

bAction	Bloco de código. Executado ao precionar o botão esquerdo do mouse.

☑ **Construtor:** New([nRow], [nCol], [cTitle], [oWnd], [bAction], [nWidth], [nHeight], [imgName], [ori], [cMsg], [btnPerLine])

☑ **Parâmetros:**

nRow	Numérico, opcional. Coordenada vertical em pixels ou carateres.
nCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres
oWnd	Objeto, opcional. Janela ou controle onde a botão deverá ser criado
bAction	Bloco de código, opcional. Bloco que deverá ser acionado quando o botão for pressionado.
nWidth	Numérico, opcional. Largura em pixels
nHeigth	Numérico, opcional. Altura em pixels.
imgName	Caracter. Imagem a ser inserida ao lado esquerdo do componente
ori	Numerico, opcional. Orientação dos botões.
cMsg	Caracter, Opicional. Descrição do tipo tooltip(hint) do item
btnPerLine	Numerico, opcional. Quantidade de botões por linha

☑ **Métodos auxiliares:**

LoadImage

- **Descrição:** Muda a figura a esquerda do componente
- **Sintaxe:** LoadImage(cImagem)
- **Parâmetros:**

cImagem	Nome da imagem que será carregada pelo objeto

- **Retorno:**

Nenhum	-

AddButton

- **Descrição:** Insere um botão
- **Sintaxe:** AddButton(cTexto)
- **Parâmetros:**

cTexto	Texto para exibição no botão inserido.

- **Retorno:**

Nenhum	-

SetTitle

- **Descrição:** Muda o titulo e o numero de botões por linha
- **Sintaxe:** SetTitle(cTexto, nBotoes)
- **Parâmetros:**

cTexto	Texto que será atribuído aos botões
nBotoes	Número de botões por linha

- **Retorno:**

Nenhum	-

SetFonts

- **Descrição:** Muda a font do titulo e dos botões
- **Sintaxe:** SetTitle(cFontTit, nSizeTit, nFontBut, nSizeBut)
- **Parâmetros:**

cFontTit	Nome da fonte que será atribuída ao título
nSizeTit	Tamanho da fonte do título
nFontBut	Nome da fonte que será atribuída aos botões
nSizeBut	Tamanho da fonte dos botões

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 304,504 PIXEL TITLE 'Exemplo'

      otMultiBtn := tMultiBtn():New( 01,01,'cTitle1',oDlg,;
                   {|x|Alert(Str(x)) },200,150,'afastamento',0,'cTitle',3
)

      otMultiBtn:SetFonts('Tahoma',16,'Tahoma',10)
      otMultiBtn:AddButton('btn01')
      otMultiBtn:AddButton('btn02')
      otMultiBtn:AddButton('btn04')
      otMultiBtn:AddButton('btn05')

      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
.
```

TMULTIGET()

☑ **Descrição:** Classe de objetos visuais do tipo controle - edição de texto de múltiplas linhas.

☑ **Propriedades:**

lWordWrap	Lógico. Se .T., faz quebra automática de linhas.

☑ **Construtor:** New([anRow], [anCol], [abSetGet], [aoWnd], [anWidth], [anHeight], [aoFont], [alHScroll], [anClrFore], [anClrBack], [oPar11], [alPixel], [cPar13], [lPar14], [abWhen], [lPar16], [lPar17], [alReadOnly], [abValid], [bPar20], [lPar21], [alNoBorder], [alNoVScroll])

☑ **Parâmetros:**

Parâmetro	
anRow	Numérico, opcional. Coordenada vertical em pixels ou caracteres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
abSetGet	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.
aoWnd	Objeto, opcional. Janela ou controle onde o controle será criado.
anWidth	Numérico, opcional. Largura do controle em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
aoFont	Objeto, opcional. Objeto tipo tFont utilizado para definir as características da fonte utilizada para exibir o conteúdo do controle.
alHScroll	Lógico, opcional. Se .T., habilita barra de rolagem horizontal.
anClrFore	Numérico, opcional. Cor de fundo do controle.
anClrBack	Numérico, opcional. Cor do texto do controle.
oPar11	Reservado.
alPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
cPar13	Reservado.
lPar14	Reservado.
abWhen	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.
lPar16	Reservado.
lPar17	Reservado.
alReadOnly	Lógico, opcional. Se .T. o controle so permitira leitura.
abValid	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.
bPar20	Reservado.
lPar21	Reservado.

alNoBorder	Lógico, opcional. Se .T. cria controle sem borda.
alNoVScroll	Lógico, opcional. Se .T., habilita barra de rolagem vertical.

☑ **Métodos auxiliares:**

EnableVScroll

- **Descrição:** Habilita a barra de rolagem vertical.
- **Sintaxe:** EnableVScroll(lEnable)
- **Parâmetros:**

lEnable	Lógico, obrigatório. Se .T. habilita se .F. desabilita a barra de rolagem.

- **Retorno:**

Nenhum	-

EnableHScroll

- **Descrição:** Habilita a barra de rolagem horizontal.
- **Sintaxe:** EnableHScroll(lEnable)
- **Parâmetros:**

lEnable	Lógico, obrigatório. Se .T. habilita se .F. desabilita a barra de rolagem.

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
#include "protheus.ch"
User Function Teste()

Local oDlg, oMemo, cMemo:= space(50)

DEFINE MSDIALOG oDlg FROM 0,0 TO 400,400 PIXEL TITLE "My test"

oMemo:= tMultiget():New(10,10,{|u|if(Pcount()>0,cMemo:=u,cMemo)};
   ,oDlg,100,100,,,,,,.T.)

@ 200,10 BUTTON oBtn PROMPT "Fecha" OF oDlg PIXEL ACTION oDlg:End()

ACTIVATE MSDIALOG oDlg CENTERED

MsgStop(cMemo)

Return NIL
```

TOLECONTAINER()

☑ **Descrição:** Classe de objetos visuais do tipo controle, a qual permite a criação de um botão vinculado a um objeto OLE.

☑ **Propriedades:**

-	Herdadas das classes superiores.

☑ **Construtor:** New([nRow], [nCol], [nWidth], [nHeight], [oWnd], [lAutoActivate], [cFileName])

☑ **Parâmetros:**

nRow	Numérico, opcional. Coordenada vertical em pixels ou carateres.
nCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
nWidth	Numérico, opcional. Largura do botão em pixels
nHeight	Numérico, opcional. Altura do botão em pixels.
oWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
lAutoActivate	Nao utilizado
cFileName	Caracter. Endereço do arquivo Ole a ser aberto

☑ **Métodos auxiliares:**

OpenFromFile

- **Descrição:** Abre arquivo do tipo OLE.
- **Sintaxe:** New([cFileName, [lAsIcon], [allowInPlace])
- **Parâmetros:**

cFileName	Caracter. Endereço do arquivo Ole a ser aberto
lAsIcon	Logico. Define objeto Ole como Icone
allowInPlace	Logico opcional. Permite abertura de arquivo local

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 304,504 PIXEL TITLE 'Exemplo'

      // TOleContainer
      oTOleContainer := TOleContainer():New(
05,05,306,134,oDlg,.T.,'C:\Lixo\TesteRemote.doc' )

      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

TPAGEVIEW()

☑ **Descrição:** Classe de objetos visuais do tipo controle, que permite a visualização de arquivos no formato gerado pelo spool de impressão do Protheus.

☑ **Propriedades:**

-	Herdadas das classes superiores.

☑ **Construtor:** New([nCol], [nRow], [nWidth], [nHeight], [oPrinter], [oWnd], [nPageWidth], [nPageHeight])

☑ **Parâmetros:**

nCol	Numérico, opcional. Coordenada vertical em pixels ou carateres.
nRow	Numérico, opcional. Coordenada horizontal em pixels ou caracteres
nWidth	Numérico, opcional. Largura em pixels
nHeigth	Numérico, opcional. Altura em pixels.
oPrinter	Objeto do tipo TMsPrinter()
oWnd	Objeto, opcional. Janela ou controle onde a botão deverá ser criado
nPageWidth	Numérico, opcional. Largura em pixels da pagina
nPageHeight	Numérico, opcional. Altura em pixels da pagina

☑ **Métodos auxiliares:**

PageSize

- **Descrição:** Define o tamanho da pagina.
- **Sintaxe:** AddItem(nLargura, nAltura)
- **Parâmetros:**

nLargura	Lagura da página
nAltura	Altura da página

- **Retorno:**

Nenhum	-

PrintPage

- **Descrição:** Imprime uma determinada página.
- **Sintaxe: PrintPage(**nNrPagina**)**
- **Parâmetros:**

nNrPagina	Página que será impressa

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 304,504 PIXEL TITLE 'Exemplo'

      // TPageView
      __RelDir := WSPLRelDir()
      oPrinter := TMsPrinter():New()
      oPrinter:SetFile(__RELDIR + 'matr020.prt',.F.)
      oTPageView := TPageView():New(
0,0,500,300,oPrinter,oTFolder2:aDialogs[07],550,350 )
      oTPageView:Reset(400,400)
      oTPageView:PrintPage(1)
      oTPageView:Show()
      oTPageView:nZoom := 150

      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

TPANEL()

☑ **Descrição:** 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.

☑ **Propriedades:**

-	Herdadas das classes superiores

☑ **Construtor:** New([anRow], [anCol], [acText], [aoWnd], [aoFont], [alCentered], [lPar6], [anClrText], [anClrBack], [anWidth], [anHeight], [alLowered], [alRaised])

☑ **Parâmetros:**

anRow	Numérico, opcional. Coordenada vertical em pixels.
anCol	Numérico, opcional. Coordenada horizontal em pixels.
acText	Caractere, opcional. Texto a ser exibido ao fundo.
aoWnd	Objeto, opcional. Janela ou controle onde será criado o objeto.
alCentered	Lógico, opcional. Se .T. exibe o texto de título ao centro do controle.
lPar6	Reservado.
anClrText	Numérico, opcional. Cor do texto do controle.
anClrBack	Numérico, opcional. Cor do fundo do controle.
anWidth	Numérico, opcional. Largura do controle em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
alLowered	Lógico, opcional. Se .T. exibe o painel rebaixado em relação ao controle de fundo.
alRaised	Lógico, opcional. Se .T. exibe a borda do controle rebaixada em relação ao controle de fundo.

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg, oButton, oCombo, cCombo, cGet1:='Teste'
DEFINE MSDIALOG oDlg FROM 0,0 TO 300,300 PIXEL TITLE 'Meu test'
 oPanel:= tPanel():New(10,10,"",oDlg,,,,,CLR_BLUE,100,100) // cria o
painel
 @ 10,10 BUTTON oBtn1 PROMPT 'hide' OF oPanel ACTION oPanel:Hide() // cria
botão sobre o painel
 @ 200,10 BUTTON oBtn2 PROMPT 'show' OF oDlg ACTION oPanel:Show() // cria
botão fora o painel
ACTIVATE MSDIALOG oDlg CENTERED
Return
```

TRADMENU()

☑ **Descrição:** Classe de objetos visuais do tipo controle – TRadMenu, a qual permite criar um controle visual no formato Radio Button.

☑ **Propriedades:**

bChange	Bloco de código disparado na mudança de item selecionado
bSetGet	Bloco de código disparado na mudança de item selecionado, responsavel pela mudança de valor da variavel numerica que indica o item selecionado.
bWhen	Bloco de código que permite ou não a alteração do objeto
bValid	Bloco de código executado na saida do objeto.

☑ **Construtor:** New([nRow], [nCol], [aItems], [bSetGet], [oDlg], [aHelpIds], [bChange], [nClrText], [nClrPane], [cMsg], [lUpdate], [bWhen], [nWidth], [nHeight], [bValid], [lDesign], [l3d], [lPixel])

☑ **Parâmetros:**

nRow	Numérico, opcional. Coordenada vertical
nCol	Numérico, opcional. Coordenada horizontal
aItems	Vetor, elementos do Radio
bSetGet	Code-block, Responsável pela setagem de valor
oDlg	Objeto, opcional. Janela ou controle onde o divisor deverá ser criado.
aHelpIds	Não utilizado
bChange	Bloco de código, opcional. Executado quando o item selecionado é alterado.
nClrText	Numérico, opcional. Cor do texto da janela.
nClrPane	Numérico, opcional. Cor de fundo da janela.
cMsg	Caracter, opcional. Mesamge ao posicionar o mouse sobre o objeto
lUpdate	Não utilizado
bWhen	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.

nWidth	Numérico, opcional. Largura do objeto
nHeight	Numérico, opcional. Altura do objeto
bValid	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.
lDesign	Não utilizado
l3d	Não utilizado
lPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.

☑ **Métodos auxiliares:**

SetOption

- **Descrição:** Seleciona um item.
- **Sintaxe: SetOption(nItem)**
- **Parâmetros:**

nItem	Item que será selecionado

- **Retorno:**

Nenhum	-

EnableItem

- **Descrição:** Habilita/Desabilita um Item
- **Sintaxe: EnableItem(nItem, lStatus)**
- **Parâmetros:**

nItem	Item que será modificado
lStatus	Status que será atribuído (.T. – Enable , .F. – Disable)

- **Retorno:**

Nenhum	-

Enable

- **Descrição:** Habilita um item.
- **Sintaxe: Enable(nItem)**
- **Parâmetros:**

nItem	Item que será habilitado

- **Retorno:**

Nenhum	-

Disable

- **Descrição:** Desabilita um item
- **Sintaxe: Disable(nItem)**
- **Parâmetros:**

nItem	Item que será desabilitado

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Anotações

Exemplo:

```
#include 'protheus.ch'
user function TRadMenu()
      DEFINE MSDIALOG oDlg FROM 0,0 TO 270,400 PIXEL TITLE 'Exemplo da
TCBrowse'
            // Variavel numerica que guarda o item selecionado do Radio
            nRadio := 1
            // Cria o Objeto
            oRadio := TRadMenu():New
(01,01,{'Item01','Item02','Item03','Item04','Item05'},;

,oDlg,,,,,,,,200,200,,,,.T.)
            // Seta Eventos
            oRadio:bchange    := {|| Alert('bChange') }
            oRadio:bSetGet    := {|u|Iif (PCount()==0,nRadio,nRadio:=u)}
            oRadio:bWhen            := {|| .T. }
            oRadio:bValid     := {|| Alert('bValid') }
            // Principais comandos
            oBtn  := TButton():New( 060, 001, 'Retorna item selecionado',;
            oDlg,{|| Alert(nRadio)             }, 120,
010,,,.F.,.T.,.F.,,.F.,,,.F. )
            oBtn  := TButton():New( 070, 001, 'SetOption(2) (Seta um
item)',;
            oDlg,{|| oRadio:SetOption(2)       }, 120,
010,,,.F.,.T.,.F.,,.F.,,,.F. )
            oBtn  := TButton():New( 080, 001, 'EnableItem(2,.T.) (Habilita
item)',;
            oDlg,{|| oRadio:EnableItem(2,.T.)  }, 120,
010,,,.F.,.T.,.F.,,.F.,,,.F. )
            oBtn  := TButton():New( 090, 001, 'EnableItem(2,.F.)
(Desabilita item)',;
            oDlg,{|| oRadio:EnableItem(2,.F.)  }, 120,
010,,,.F.,.T.,.F.,,.F.,,,.F. )
            oBtn  := TButton():New( 100, 001, 'Enable(3) (Habilita
item)',;
            oDlg,{|| oRadio:Enable(3)          }, 120,
010,,,.F.,.T.,.F.,,.F.,,,.F. )
            oBtn  := TButton():New( 110, 001, 'Disable(3) (Desabilita
item)',;
            oDlg,{|| oRadio:Disable(3)         }, 120,
010,,,.F.,.T.,.F.,,.F.,,,.F. )

      ACTIVATE MSDIALOG oDlg CENTERED
Return
```

TSBROWSE()

☑ **Descrição:** Classe de objetos visuais do tipo controle – TSBrowse, a qual permite criar um controle visual do tipo Grid.

☑ **Propriedades:**

+	Herdadas da classe superior
nAt	Linha atualmente selecionada / posicionada no objeto
nLen	Número total de linhas do objeto

☑ **Construtor:** New([nRow], [nCol], [nWidth], [nHeight], [oWnd], [bChange], [nHWidth], [oFont], [nLines])

☑ **Parâmetros:**

nRow	Numérico, opcional. Coordenada vertical
nCol	Numérico, opcional. Coordenada horizontal
nWidth	Numérico, opcional. Largura do objeto
nHeight	Numérico, opcional. Altura do objeto
oWnd	Objeto, opcional. Janela ou controle onde o divisor deverá ser criado.
bChange	Bloco de código, na mudança de linha
nHWidth	Não utilizado
oFont	Objeto, opcional. Fonte
nLines	Númerico. Nr de linhas por célula

☑ **Métodos auxiliares:**

GoUp

- ❑ **Descrição:** Salta uma linha para cima.
- ❑ **Sintaxe: GoUp**( )
- ❑ **Parâmetros:**

Nenhum	-

- ❑ **Retorno:**

Nil	

GoDown

- ❑ **Descrição:** Salta uma linha para baixo.
- ❑ **Sintaxe: GoDown**( )
- ❑ **Parâmetros:**

Nenhum	-

- ❑ **Retorno:**

Nil	

GoTop

- **Descrição:** Salta para primeira linha.
- **Sintaxe: GoTop**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoBottom

- **Descrição:** Salta para ultima linha.
- **Sintaxe: GoBottom**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

RowCount

- **Descrição:** Retorna numero de linhas visiveis.
- **Sintaxe: RowCount**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

LEditCell

- **Descrição:** Edita o valor de uma coluna.
- **Sintaxe:** lEditCell(aList, oList, cPicture, nCol)
- **Parâmetros:**

aList	Vetor da Browse onde estão os valores da mesma
oList	Objeto, Browse a ser editado
cPicture	Caracter, picture necessária para edição do campo
nCol	Numérico, coluna a ser editada.

- **Retorno:**

Nil	

☑ **Aparência:**

Anotações

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 302,402 PIXEL TITLE 'Exemplo'

      aBrowse   := {{'CLIENTE 001','RUA CLIENTE 001','BAIRRO CLIENTE
001'},;
                    {'CLIENTE 001','RUA CLIENTE 001','BAIRRO CLIENTE
001'},;
                    {'CLIENTE 001','RUA CLIENTE 001','BAIRRO CLIENTE 001'}
}
      oBrowse := TSBrowse():New(01,01,245,150,oDlg,,16,,5)
      nForeCor := CLR_GRAY
      nBackCor := CLR_WHITE
      oBrowse:AddColumn( TcColumn():New('Nome',,,{|| nForeCor },{||
nBackCor }) )
      oBrowse:AddColumn( TcColumn():New('Endereço',,,{|| nForeCor },{||
nBackCor }) )
      oBrowse:AddColumn( TcColumn():New('Bairro',,,{|| nForeCor },{||
nBackCor }) )
      oBrowse:SetArray(aBrowse)

      // Principais commandos
      TButton():New(160,001,'GoUp()',oDlg,{|| oBrowse:GoUp()
},40,10,,,,.T.)
      TButton():New(170,001,'GoDown()',oDlg,{|| oBrowse:GoDown()
},40,10,,,,.T.)
      TButton():New(180,001,'GoTop()',oDlg,{|| oBrowse:GoTop()
},40,10,,,,.T.)
      TButton():New(190,001,'GoBottom()' , oDlg,{|| oBrowse:GoBottom()
},40,10,,,,.T.)
      TButton():New(160,060,'nAt (Linha selecionada)',oDlg,;
                          {|| Alert (oBrowse:nAt)},80,10,,,,.T.)
      TButton():New(170,060,'nRowCount (Nr de linhas visiveis)',oDlg,;
                          {|| Alert(oBrowse:nRowCount()) },80,10,,,,.T.)
      TButton():New(180,060, 'nLen (Numero total de linhas)', oDlg,;
                          {|| Alert(oBrowse:nLen) },80,10,,,,.T.)
       TButton():New( 190, 060, 'lEditCell (Edita a celula)', oDlg,;
                          {|| lEditCell(@aBrowse,oBrowse,'@!',3)
},40,10,,,,.T.)


      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL

.
```

TSAY()

☑ **Descrição:** Classe de objetos visuais do tipo controle – tSay, a qual exibe o conteúdo de texto estático sobre uma janela ou controle previamente definidos.

☑ **Propriedades:**

lWordWrap	Lógico. Se .T. quebra o texto em várias linhas de maneira a enquadrar o conteúdo na área determinada para o controle, sendo o padrão .F.
lTransparent	Lógico. Se .T. a cor de fundo do controle é ignorada assumindo o conteúdo ou cor do controle ou janela ao fundo, sendo o padrão .T.

☑ **Construtor:** New([anRow], [anCol], [abText], [aoWnd], [acPicture], [aoFont], [lPar7], [lPar8], [lPar9], [alPixels], [anClrText], [anClrBack], [anWidth], [anHeight], [lPar15], [lPar16], [lPar17], [lPar18], [lPar19])

☑ **Parâmetros:**

anRow	Numérico, opcional. Coordenada vertical em pixels ou caracteres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
abText	Codeblock, opcional. Quando executado deve retornar uma cadeia de caracteres a ser exibida.
aoWnd	Objeto, opcional. Janela ou diálogo onde o controle será criado.
acPicture	Caractere, opcional. Picture de formatação do conteúdo a ser exibido.
aoFont	Objeto, opcional. Objeto tipo tFont para configuração do tipo de fonte que será utilizado para exibir o conteúdo.
lPar7	Reservado.
lPar8	Reservado.
lPar9	Reservado.
alPixels	Lógico, opcional. Se .T. considera coordenadas passadas em pixels se .F., padrão, considera as coordenadas passadas em caracteres.
anClrText	Numérico, opcional. Cor do conteúdo do controle.
anClrBack	Numérico, opcional. Cor do fundo do controle.
anWidth	Numérico, opcional. Largura do controle em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
lPar15	Reservado.
lPar16	Reservado.
lPar17	Reservado.
lPar18	Reservado.
lPar19	Reservado.

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg, oButton, oCombo, cCombo, cGet1:='Teste'
DEFINE MSDIALOG oDlg FROM 0,0 TO 300,300 PIXEL TITLE 'Meu test'
 oSay:= tSay():New(01,01,{||'para exibir'},oDlg,,,,;
    ,,.T.,CLR_RED,CLR_WHITE,100,20)
ACTIVATE MSDIALOG oDlg CENTERED
Return
```

Anotações

TSCROLLBOX()

☑ **Descrição:** Classe de objetos visuais do tipo controle – tScrollbox, a qual permite criar um painel com scroll deslizantes nas laterais (horizontais e verticais) do controle.

☑ **Propriedades:**

-	Herdadas das classes superiores

☑ **Construtor:** New([aoWnd], [anTop], [anLeft], [anHeight], [anWidth], [alVertical], [alHorizontal], [alBorder])

☑ **Parâmetros:**

aoWnd	Objeto, opcional. Janela ou controle onde o controle será criado.
anTop	Numérico, opcional. Coordenada vertical em pixels.
anLeft	Numérico, opcional. Coordenada horizontal em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
anWidth	Numérico, opcional. Largura do controle em pixels.
alVertical	Lógico, opcional. Se .T. exibe a barra de scroll vertical.
alHorizontal	Lógico, opcional. Se .T. exibe a barra de scroll horizontal.
alBorder	Lógico, opcional. Se .T. exibe a borda do controle.

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg, oScr, oGet1, oGet2, oGet3
Local cGet1, cGet2, cGet3
cGet1:= Space(10)
cGet2:= Space(10)
cGet3:= Space(10)
DEFINE MSDIALOG oDlg FROM 0,0 TO 220,220 PIXEL TITLE 'Meu test'
 oScr:= TScrollBox():New(oDlg,10,10,100,100,.T.,.T.,.T.) // cria controles
dentro do scrollbox
 @ 10,10 MSGET oGet1 VAR cGet1 SIZE 100,10 OF oScr PIXEL
 @ 50,10 MSGET oGet2 VAR cGet2 SIZE 100,10 OF oScr PIXEL
 @ 150,100 MSGET oGet3 VAR cGet3 SIZE 100,10 OF oScr PIXEL
ACTIVATE MSDIALOG oDlg CENTERED
Return
```

TSIMPLEEDITOR()

☑ **Descrição:** 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®

☑ **Propriedades:**

-	Herdadas das classes superiores.

☑ **Construtor:** New([anRow], [anCol], [aoWnd], [anWidth], [anHeight],[acText], [alReadOnly])

☑ **Parâmetros:**

anRow	Numérico, opcional. Coordenada vertical em pixels ou carateres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
aoWnd	Janela ou controle onde o editor deverá ser criado.
anWidth	Numérico, opcional. Largura do editor em pixels.
anHeight	Numérico, opcional. Altura do editor em pixels.
acText	Texto, opcional. Inserido na inicialização do objeto.
alReadOnly	Lógico, opcional. Permite ou não a edição do texto

☑ **Métodos auxiliares:**

Create

- **Descrição:** Método construtor opcional da classe.
- **Sintaxe:** Create(aoWnd)
- **Parâmetros:**

aoWnd	Objeto, opcional. Janela ou controle onde o editor deverá ser criado.

- **Retorno:**

Objeto	Objeto do tipo TsimpleEditor.

Load

- **Descrição:** Carrega um texto para o editor.
- **Sintaxe:** Load(acTexto)
- **Parâmetros:**

acTexto	Texto. Texto que inicializará o editor.

- **Retorno:**

Nenhum	-

TextBold

- **Descrição:** Texto em Negrito.
- **Sintaxe:** TextBold(alBold)
- **Parâmetros:**

alBold	Lógico. Habilita ou Desabilita a sessão do texto como Negrito.

- **Retorno:**

Nenhum	-

TextUnderline

- **Descrição:** Texto Sublinhado.
- **Sintaxe:** TextUnderline(alUnderline)
- **Parâmetros:**

alUnderline	Lógico. Habilita ou Desabilita a sessão do texto Sublinhado.

- **Retorno:**

Nenhum	-

TextItalic

- **Descrição:** Texto Itálico.
- **Sintaxe:** TextItalic(alItalic)
- **Parâmetros:**

alItalic	Lógico. Habilita ou Desabilita a sessão do texto Itálico.

- **Retorno:**

Nenhum	-

TextFamily

- **Descrição:** Família de fontes.
- **Sintaxe:** TextFamily(acFamily)
- **Parâmetros:**

acFamily	Texto. Nome da família da fonte a ser usada na sessão do texto.

- **Retorno:**

Nenhum	-

TextSize

- **Descrição:** Tamanho da fonte.
- **Sintaxe:** TextSize(anSize)
- **Parâmetros:**

anSize	Numérico. Tamanho da fonte utilizada na sessão do texto.

- **Retorno:**

Nenhum	-

TextStyle

- **Descrição:** Estilo do parágrafo.
- **Sintaxe:** TextStyle(anStyle)
- **Parâmetros:**

anStyle	Numérico. Estilo do parágrafo a ser utilizada na sessão do texto. 1 – Normal 2 – Disco (Bullet) 3 – Circulo (Bullet) 4 – Quadrado (Bullet) 5 – Ordem decimal 6 – Ordem alfanumérica minúsculo 7 – Ordem alfanumérica maiúsculo

- **Retorno:**

Nenhum	-

TextAlign

- **Descrição:** Alinhamento do texto.
- **Sintaxe:** TextAlign(anAlign)
- **Parâmetros:**

anAlign	Numérico. Tipo do alinhamento do parágrafo. 1 – Esquerda 2 – Direita 3 – Centralizado 4 – Justificado

- **Retorno:**

Nenhum	-

TextFormat

- **Descrição:** Formato do texto.
- **Sintaxe:** TextFormat(anFormat)
- **Parâmetros:**

anFormat	Numérico. Formato do texto 1 – Html 2 – Plain Text

- **Retorno:**

Nenhum	-

RetText

- **Descrição:** Retorna o texto em formato string.
- **Sintaxe:** RetText(void)
- **Parâmetros:**

void	Retorna uma string com o conteúdo do editor

- **Retorno:**

Nenhum	-

RetTextSel

- **Descrição:** Retorna o texto selecionado em formato string..
- **Sintaxe:** RetText()
- **Parâmetros:**

Nenhum	-

- **Retorno:**

String	Texto selecionado.

TextStatus

- **Descrição:** Retorna um array com as propriedades do texto posicionado.
- **Sintaxe:** TextStatus()
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Array	Retorna um array com as seguintes propriedades : 1 – Lógico Negrito 2 – Lógico Itálico 3 – Lógico Sublinhado 4 – Caracter RGB da cor 5 – Caracter Tamanho da fonte 6 – Caracter Fonte

☑ **Aparência:**

Exemplo:

```
#include "protheus.ch"


User Function MyEditor()
Local oDlg, oEditor
DEFINE MSDIALOG oDlg FROM 0,0 TO 500,600 PIXEL TITLE "Meu Editor"
// Usando o método create
oEdit := tSimpleEditor():Create( oDlg )
oEdit:nTop      := 10
oEdit:nLeft     := 10
oEdit:nWidth    := 600
oEdit:nHeight   := 500


// Usando o método new
oEdit := tSimpEdit():New( 0, 0, oDlg, 500, 600 )


ACTIVATE MSDIALOG oDlg CENTERED
Return Nil
```

TSLIDER()

☑ **Descrição:** Classe de objetos visuais do tipo controle – tSlider, a qual permite criar um controle visual do tipo botão deslizante.

☑ **Propriedades:**

bChange	Bloco de código. Executado toda vez que o valor é alterado retornando o novo valor.

☑ **Construtor:** New([anRow], [anCol], [aoWnd], [abChange], [anWidth], [anHeight], [acMsg], [abWhen])

☑ **Parâmetros:**

anRow	Numérico, opcional. Coordenada vertical em pixels
anCol	Numérico, opcional. Coordenada horizontal em pixels
aoWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
abChange	Bloco de código, opcional. Bloco que deverá ser acionado quando o botão for movimentado.
anWidth	Numérico, opcional. Largura do botão em pixels.
anHeight	Numérico, opcional. Altura do botão em pixels.
acMsg	Caracter, opcional. Mensagem de hint do botão
abWhen	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.

☑ **Métodos auxiliares:**

Create

- **Descrição:** Método construtor da classe.
- **Sintaxe:** Create(aoWnd)
- **Parâmetros:**

aoWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.

- **Retorno:**

Objeto	Objeto do tipo TSlide criado.

setRange

- **Descrição:** Especifica a faixa de valores.
- **Sintaxe:** setRange(min, max)
- **Parâmetros:**

Min	Númérico. Valor mínimo do botão.
Max	Númérico. Valor máximo do botão.

- **Retorno:**

Nenhum	-

setMarks

- **Descrição:** Especifica o tipo de marcação do botão.
- **Sintaxe:** setMarks(nTipo)
- **Parâmetros:**

nTipo	Númérico. Tipo de marcação do botão. 1 -
Max	Númérico. Valor máximo do botão. 0 – Sem marcação 1 – Acima (esquerda se vertical) 2 – Abaixo (direita se vertical) 3 – Ambos os lados

- **Retorno:**

Nenhum	-

setInterval

- **Descrição:** Especifica a distância entre um marcador e outro.
- **Sintaxe:** setInterval(nInterval)
- **Parâmetros:**

nInterval	Númérico. Valor entre os marcadores

- **Retorno:**

Nenhum	-

setValue

- **Descrição:** Especifica um valor para o botão.
- **Sintaxe:** setValue(nVal)
- **Parâmetros:**

nVal	Númérico. Valor do botão

- **Retorno:**

Nenhum	-

setStep

- **Descrição:** Especifica o valor dos passos do botão.
- **Sintaxe:** setStep(nStep)
- **Parâmetros:**

nStep	Númérico. Valor do passo do botão

- **Retorno:**

Nenhum	-

setOrient

- **Descrição:** Especifica a orientação do botão, horizontal ou vertical.
- **Sintaxe:** setOrient(nOrient)
- **Parâmetros:**

nOrient	Númérico. Orientação do botão 0 – Horizontal 1 – Vertical

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
#include “protheus.ch”
#include “hbutton.ch”
User Function MytSlider()
Local oDlg, oSlider
DEFINE MSDIALOG oDlg FROM 0,0 TO 500,600 PIXEL TITLE “Meu tSlider”
// Usando o método create
oSlider:= tSlider():Create( oDlg )
oSlider:nTop      := 100
oSlider:nLeft     := 10
oSlider:nWidth    := 100
oSlider:nHeight   := 30


// Usando o command
@ 100, 10 SLIDER oSlider SIZE 30, 100 OF oDlg MESSAGE 'tSlider'
ACTIVATE MSDIALOG oDlg CENTERED
Return Nil
```

TSPLITTER()

☑ **Descrição:** Classe de objetos visuais do tipo controle – tSplitter, a qual permite criar um controle visual do tipo divisor.

☑ **Propriedades:**

-	Herdadas as classes superiores.

☑ **Construtor:** New([anRow], [anCol], [aoWnd], [anWidth], [anHeight], [anOrientation])

☑ **Parâmetros:**

anRow	Numérico, opcional. Coordenada vertical
anCol	Numérico, opcional. Coordenada horizontal
aoWnd	Objeto, opcional. Janela ou controle onde o divisor deverá ser criado.
anWidth	Numérico, opcional. Largura do objeto
anHeight	Numérico, opcional. Altura do objeto
anOrientation	Numérico, opcional. Sentido no qual deverão ser criado os divisores. 0 – Horizontal 1 – Vertical

☑ **Métodos auxiliares:**

Create

- **Descrição:** Método construtor da classe.
- **Sintaxe:** Create(aoWnd)
- **Parâmetros:**

aoWnd	Objeto, opcional. Janela ou controle onde o divisor deverá ser criado.

- **Retorno:**

Objeto	Objeto do tipo Tsplitter gerado.

setOrient

- **Descrição:** Especifica a orientação do divisor, horizontal ou vertical.
- **Sintaxe:** setOrient(nOrient)
- **Parâmetros:**

nOrient	Númérico. Orientação do botão 0 – Horizontal 1 – Vertical

- **Retorno:**

Nenhum	-

setChildCollapse

- **Descrição:** Especifica se os elementos podem ser “collapsibles”.
- **Sintaxe:** setChildCollpse(lColl)
- **Parâmetros:**

lColl	Lógico. Ativa / Desativa

- **Retorno:**

Nenhum	-

setCollapse

- **Descrição:** Especifica o objeto que pode ser “collapsible”.
- **Sintaxe:** setCollapse(oObj, lColl)
- **Parâmetros:**

oObj	Objeto. Controle a ser “collapsed”
lColl	Lógico. Ativa / Desativa

- **Retorno:**

Nenhum	-

movToLast

- **Descrição:** Coloca o objeto como ultimo das divisões.
- **Sintaxe:** movToLast(oObj)
- **Parâmetros:**

oObj	Objeto. Controle a posicionado

- **Retorno:**

Nenhum	-

movToFirst

- **Descrição:** Coloca o objeto como primeiro das divisões.
- **Sintaxe:** movToFirst(oObj)
- **Parâmetros:**

oObj	Objeto. Controle a ser posicionado

- **Retorno:**

Nenhum	-

setOpaqueResize

- **Descrição:** Especifica se o resize deve ser opaco.
- **Sintaxe:** setOpaqueResize(lOpaq)
- **Parâmetros:**

lOpaq	Lógico. Ativa / Desativa

- **Retorno:**

Nenhum	-

setResizeMode

- **Descrição:** Especifica o mode de “resize” do objeto..
- **Sintaxe: setResizeMode** (oObj, nMode)
- **Parâmetros:**

oObj	Objeto. Controle a ser “resized”
nMode	Numérico. Modo do “resize” 0 – Stretch 1- KeepSize 2 – FollowSizeHint 3 – Auto

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Anotações

Exemplo:

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

User Function MytSplitter()
Local oDlg, oSplitter, oPanel1, oPanel2, oPanel3


DEFINE MSDIALOG oDlg FROM 0,0 TO 500,600 PIXEL TITLE "Meu tSplitter"

// Usando o método create
oSplitter := tSplitter():Create( oDlg )
@ 1, 1 MSPANEL oPanel1 OF oSplitter
@ 1, 2 MSPANEL oPanel2 OF oSplitter
@ 1, 3 MSPANEL oPanel3 OF oSplitter

// Usando o command
@ 1, 1 SPLITTER oSplitter SIZE 100, 100 OF oDlg
@ 1, 1 MSPANEL oPanel1 OF oSplitter
@ 1, 2 MSPANEL oPanel2 OF oSplitter
@ 1, 3 MSPANEL oPanel3 OF oSplitter

ACTIVATE MSDIALOG oDlg CENTERED

Return Nil
```

TTABS()

☑ **Descrição:** Classe de objetos visuais do tipo controle – TTabs, a qual permite criar um controle visual do tipo pasta.

☑ **Propriedades:**

-	Herdadas das classes superiores.

☑ **Construtor:** New([anTop], [anLeft], [aPrompts], [bAction], [oWnd], [nOption], [nClrFore], [nClrBack], [lPixel], [lDesign], [nWidth], [nHeigth], [cMsg])

☑ **Parâmetros:**

anTop	Numérico, opcional. Coordenada vertical em pixels ou carateres.
anLeft	Numérico, opcional. Coordenada horizontal em pixels ou caracteres
aPrompts	Array, Titulo dos folders
bAction	Bloco de codigo, Disparado na troca da aba.
oWnd	Objeto, opcional. Janela ou controle onde a botão deverá ser criado
nOption	Numérico, opcional. Folder selecionado
nClrFore	Numérico, opcional. Cor de frente
nClrBack	Numérico, opcional. Cor de fundo
lPixel	Lógico, opcional. Utiliza coordenadas em pixel

lDesign	Lógico, opcional. NÃO USADO
nWidth	Numérico, opcional. Largura em pixels
nHeigth	Numérico, opcional. Altura em pixels.
cMsg	Caractere, Mensagem de Hint

☑ **Métodos auxiliares:**

AddItem

- **Descrição:** Adiciona uma aba na pasta
- **Sintaxe:** AddItem(cTítulo)
- **Parâmetros:**

cTítulo	Título da aba que será adicionada

- **Retorno:**

Nenhum	-

SetOption

- **Descrição:** Seleciona a Aba
- **Sintaxe:** SetOption(nNrAba)
- **Parâmetros:**

nNrAba	Número que identifica a posição da aba que será selecionada.

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
      DEFINE MSDIALOG oDlg FROM 0,0 TO 304,504 PIXEL TITLE 'Exemplo'

        oTTabs := TTabs():New(01,01,{'Aba01','Aba02','Aba03'},;
         {||oPanel01:LVISIBLECONTROL:=(oTTabs:nOption==1)},;
         oDlg,,RGB(255,0,0),RGB(255,255,0),.T.,,120,120,)
      oPanel01 := TPanel():New( 000, 000,'',oTTabs,,,,,,100,100,,.T. )
      oPanel01:lVisibleControl := .T.
        oBtn01  := TButton():New( 01,01,'TButton01',oPanel01,;
                 {||oTTabs:SetOption(2)}, 037,
012,,,.F.,.T.,.F.,,.F.,,,.F. )

      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

TTOOLBOX()

☑ **Descrição:** Classe de objetos visuais do tipo controle – tToolbox, a qual permite criar um controle visual para agrupar diferentes objetos.

☑ **Propriedades:**

bChangeGrp	Bloco de código. Executado na troca entre os grupos existentes.

☑ **Construtor:** New([anRow], [anCol], [aoWnd], [anWidth], [anHeight], [aoFont], [acMsg]], [abWhen])

☑ **Parâmetros:**

anRow	Numérico, opcional. Coordenada vertical em pixels ou carateres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
aoWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
anWidth	Numérico, opcional. Largura do botão em pixels.
anHeight	Numérico, opcional. Altura do botão em pixels.
aoFont	Objeto, opcional. Objeto tipo tFont com propriedades da fonte utilizada para o título do botão.
acMsg	Mensagem, opcional. Tooltip/Hint do componente.
abWhen	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.

☑ **Métodos auxiliares:**

Create

- **Descrição:** Método construtor opcional da classe.
- **Sintaxe:** Create(aoWnd)
- **Parâmetros:**

aoWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.

- **Retorno:**

Objeto	Objeto do tipo TtoolBox gerado.

addGroup

- **Descrição:** Adiciona um grupo
- **Sintaxe:** addGroup(aoObj, acName, aoIcon)
- **Parâmetros:**

aoObj	Objeto. Objeto Pai que vai ser inserido no grupo.
acName	Caractere. Descrição do grupo
aoIcon	Objeto, opcional. Ícone para o grupo

- **Retorno:**

Nenhum	-

removeGroup

- **Descrição:** Remove um grupo
- **Sintaxe:** removeGroup(aoObj)
- **Parâmetros:**

aoObj	Objeto. Objeto Pai que vai ser removido do grupo.

- **Retorno:**

Nenhum	-

setCurrentGroup

- **Descrição:** Define o grupo corrente
- **Sintaxe:** setCurrentGroup(aoObj)
- **Parâmetros:**

aoObj	Objeto. Objeto Pai que será definido como grupo corrente.

- **Retorno:**

Nenhum	-

☑ **Aparência:**

Exemplo:

```
#include "protheus.ch"

User Function MytToolbox()
Local oDlg, oTb

DEFINE MSDIALOG oDlg FROM 0,0 TO 500,600 PIXEL TITLE "Meu Toolbox"

// Usando o método create
oTb:= tToolbox():Create( oDlg )
oTb:nTop      := 100
oTb:nLeft     := 10
oTb:nWidth    := 100
oTb:nHeight   := 30

oTb:Add( oPanel, "Opção 1", oIcone)

// Usando o command
@ 200,100 TOOLBOX oTb SIZE 100,30 OF oDlg

TOOLBOX oTb ADDGROUP TITLE "Opção 1" OBJECT oPanel ICON oIcone

ACTIVATE MSDIALOG oDlg CENTERED

Return Nil
```

TWBROWSE()

☑ **Descrição:** Classe de objetos visuais do tipo controle – TWBrowse, a qual permite criar um controle visual do tipo Grid.

☑ **Propriedades:**

+	Herdadas da classe superior
nAt	Linha atualmente selecionada / posicionada no objeto
nLen	Número total de linhas do objeto

☑ **Construtor:** New([nRow], [nCol], [nWidth], [nHeigth],[bFields], [aHeaders], [aColSizes], [oDlg], [cField], [uValue1], [uValue2], [uChange],[{|nRow,nCol,nFlags|[uLDblClick]}], [{|nRow,nCol,nFlags|[uRClick]}], [oFont], [oCursor], [nClrFore], [nClrBack], [cMsg], [lUpdate], [cAlias], [lPixel], [{uWhen}], [lDesign], [bValid], [lHScroll], [lVScroll])

☑ **Parâmetros:**

nRow	Numérico, opcional. Coordenada vertical
nCol	Numérico, opcional. Coordenada horizontal
nWidth	Numérico, opcional. Largura do objeto
nHeight	Numérico, opcional. Altura do objeto
bFields	Bloco de código, Lista de Campos
aHeaders	Vetor, Descrição dos campos para no cabeçalho
aColSizes	Vetor, Largura das colunas
oDlg	Objeto, opcional. Janela ou controle onde o divisor deverá ser criado.
cField	Caracter, opcional. Campos necessários ao filtro
uValue1	Indefinido, opcional. Inicio do intervalo para o filtro
uValue2	Indefinido, opcional. Fim do intervalo para o filtro
bChange	Bloco de código, opcional. Executado quando o item selecionado é alterado.
bLDblClick	Bloco de código, opcional. Executado quando acionado duplo click do botão esquerdo do mouse sobre o controle.
bRClick	Não utilizado
oFont	Objeto, opcional. Fonte
oCursor	Objeto, opcional. Tipo do Cursor
nClrFore	Numérico, opcional. Cor do texto da janela.
nClrBack	Numérico, opcional. Cor de fundo da janela.
cMsg	Caracter, opcional. Mesamge ao posicionar o mouse sobre o objeto
lUpdate	Não utilizado
cAlias	Caracter, opcional se objeto utilizado com Vetor, obrigatorio se utilizado com Tabela
lPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
bWhen	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.
lDesign	Não Utilizado
bValid	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.
lHScroll	Lógico, opcional. Se .T., habilita barra de rolagem horizontal.
lVScroll	Lógico, opcional. Se .T., habilita barra de rolagem vertical.

☑ **Métodos auxiliares:**

GoUp

- **Descrição:** Salta uma linha para cima.
- **Sintaxe: GoUp**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoDown

- **Descrição:** Salta uma linha para baixo.
- **Sintaxe: GoDown**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoTop

- **Descrição:** Salta para primeira linha.
- **Sintaxe: GoTop**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoBottom

- **Descrição:** Salta para ultima linha.
- **Sintaxe: GoBottom**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

RowCount

- **Descrição:** Retorna numero de linhas visiveis.
- **Sintaxe: RowCount**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

LEditCell

- **Descrição:** Edita o valor de uma coluna.
- **Sintaxe:** lEditCell(aList, oList, cPicture, nCol)
- **Parâmetros:**

aList	Vetor da Browse onde estão os valores da mesma
oList	Objeto, Browse a ser editado
cPicture	Caracter, picture necessária para edição do campo
nCol	Numérico, coluna a ser editada.

- **Retorno:**

Nil	

☑ **Aparência:**

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
Local oOK := LoadBitmap(GetResources(),'br_verde')
Local oNO := LoadBitmap(GetResources(),'br_vermelho')

      DEFINE MSDIALOG oDlg FROM 0,0 TO 402,402 PIXEL TITLE 'Exemplo'

      oBrowse := TWBrowse():New( 01 , 01, 245, 150,,;
            {'','Codigo','Descrição'},{20,30,30}, oDlg, ,,,,;
            {||},,,,,,,.F.,,.T.,,.F.,,, )
       aBrowse   := {{.T.,'CLIENTE 001','RUA CLIENTE 001','BAIRRO CLIENTE
001'},;
                  {.F.,'CLIENTE 002','RUA CLIENTE 002','BAIRRO CLIENTE
002'},;
                  {.T.,'CLIENTE 003','RUA CLIENTE 003','BAIRRO CLIENTE
003'} }

      oBrowse:SetArray(aBrowse)
      oBrowse:bLine := {||{;
                If(aBrowse[oBrowse:nAt,01],oOK,oNO),;
                aBrowse[oBrowse:nAt,02],;
                aBrowse[oBrowse:nAt,03],;
                aBrowse[oBrowse:nAt,04] } }
      oBrowse:bLDblClick := ;
       {|| aBrowse[oBrowse:nAt][1] :=
!aBrowse[oBrowse:nAt][1],oBrowse:DrawSelect()}

       // Principais commandos
       TButton():New(160,001,'GoUp()',oDlg,{|| oBrowse:GoUp()
},40,10,,,,.T.)
       TButton():New(170,001,'GoDown()',oDlg,{|| oBrowse:GoDown()
},40,10,,,,.T.)
       TButton():New(180,001,'GoTop()',oDlg,{|| oBrowse:GoTop()
},40,10,,,,.T.)
       TButton():New(190,001,'GoBottom()' , oDlg,{|| oBrowse:GoBottom()
},40,10,,,,.T.)
       TButton():New(160,060,'nAt (Linha selecionada)',oDlg,;
                           {|| Alert (oBrowse:nAt)},80,10,,,,.T.)
       TButton():New(170,060,'nRowCount (Nr de linhas visiveis)',oDlg,;
                           {|| Alert(oBrowse:nRowCount()) },80,10,,,,.T.)
       TButton():New(180,060, 'nLen (Numero total de linhas)', oDlg,;
                           {|| Alert(oBrowse:nLen) },80,10,,,,.T.)
       TButton():New(190,060, 'lEditCell (Edita a celula)', oDlg,;
                           {|| lEditCell(@aBrowse,oBrowse,'@!',3)
},80,10,,,,.T.)

      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

VCBROWSE()

☑ **Descrição:** Classe de objetos visuais do tipo controle – VCBrowse, a qual permite criar um controle visual do tipo Grid.

☑ **Propriedades:**

+	Herdadas da classe superior
nAt	Linha atualmente selecionada / posicionada no objeto
nLen	Número total de linhas do objeto

☑ **Construtor:** New([nRow], [nCol], [nWidth], [nHeigth],[bFields], [aHeaders], [aColSizes], [oDlg], [cField], [uValue1], [uValue2], [uChange],[{|nRow,nCol,nFlags|[uLDblClick]}], [{|nRow,nCol,nFlags|[uRClick]}], [oFont], [oCursor], [nClrFore], [nClrBack], [cMsg], [lUpdate], [cAlias], [lPixel], [{uWhen}], [lDesign], [bValid], [lHScroll], [lVScroll])

☑ **Parâmetros:**

nRow	Numérico, opcional. Coordenada vertical
nCol	Numérico, opcional. Coordenada horizontal
nWidth	Numérico, opcional. Largura do objeto
nHeight	Numérico, opcional. Altura do objeto
bFields	Bloco de código, Lista de Campos
aHeaders	Vetor, Descrição dos campos para no cabeçalho
aColSizes	Vetor, Largura das colunas
oDlg	Objeto, opcional. Janela ou controle onde o divisor deverá ser criado.
cField	Caracter, opcional. Campos necessários ao filtro
uValue1	Indefinido, opcional. Inicio do intervalo para o filtro
uValue2	Indefinido, opcional. Fim do intervalo para o filtro
bChange	Bloco de código, opcional. Executado quando o item selecionado é alterado.
bLDblClick	Bloco de código, opcional. Executado quando acionado duplo click do botão esquerdo do mouse sobre o controle.
bRClick	Não utilizado
oFont	Objeto, opcional. Fonte
oCursor	Objeto, opcional. Tipo do Cursor
nClrFore	Numérico, opcional. Cor do texto da janela.
nClrBack	Numérico, opcional. Cor de fundo da janela.
cMsg	Caracter, opcional. Mesamge ao posicionar o mouse sobre o objeto
lUpdate	Não utilizado
cAlias	Caracter, opcional se objeto utilizado com Vetor, obrigatorio se utilizado com Tabela
lPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
bWhen	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.
lDesign	Não Utilizado

bValid	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.
lHScroll	Lógico, opcional. Se .T., habilita barra de rolagem horizontal.
lVScroll	Lógico, opcional. Se .T., habilita barra de rolagem vertical.

☑ **Métodos auxiliares:**

GoUp

- **Descrição:** Salta uma linha para cima.
- **Sintaxe: GoUp**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoDown

- **Descrição:** Salta uma linha para baixo.
- **Sintaxe: GoDown**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoTop

- **Descrição:** Salta para primeira linha.
- **Sintaxe: GoTop**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

GoBottom

- **Descrição:** Salta para ultima linha.
- **Sintaxe: GoBottom**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

RowCount

- **Descrição:** Retorna numero de linhas visiveis.
- **Sintaxe: RowCount**( )
- **Parâmetros:**

Nenhum	-

- **Retorno:**

Nil	

LEditCell

- **Descrição:** Edita o valor de uma coluna.
- **Sintaxe:** lEditCell(aList, oList, cPicture, nCol)
- **Parâmetros:**

aList	Vetor da Browse onde estão os valores da mesma
oList	Objeto, Browse a ser editado
cPicture	Caracter, picture necessária para edição do campo
nCol	Numérico, coluna a ser editada.

- **Retorno:**

Nil	

☑ **Aparência:**

Anotações

Exemplo:

```
#include 'protheus.ch'
User Function Teste()
Local oDlg
Local oOK := LoadBitmap(GetResources(),'br_verde')
Local oNO := LoadBitmap(GetResources(),'br_vermelho')
      DEFINE MSDIALOG oDlg FROM 0,0 TO 402,402 PIXEL TITLE 'Exemplo'

      oBrowse := VCBrowse():New( 01 , 01, 245, 150,,;
            {'','Codigo','Descrição'},{20,30,30}, oDlg, ,,,,;
            {||},,,,,,,.F.,,.T.,,.F.,,, )
      aBrowse   := {{.T.,'CLIENTE 001','RUA CLIENTE 001','BAIRRO CLIENTE
001'},;
                  {.F.,'CLIENTE 002','RUA CLIENTE 002','BAIRRO CLIENTE
002'},;
                  {.T.,'CLIENTE 003','RUA CLIENTE 003','BAIRRO CLIENTE
003'} }

      oBrowse:SetArray(aBrowse)
      oBrowse:bLine := {||{;
                If(aBrowse[oBrowse:nAt,01],oOK,oNO),;
                aBrowse[oBrowse:nAt,02],;
                aBrowse[oBrowse:nAt,03],;
                aBrowse[oBrowse:nAt,04] } }
      oBrowse:bLDblClick :=;
        {|| aBrowse[oBrowse:nAt][1] :=
!aBrowse[oBrowse:nAt][1],oBrowse:DrawSelect()}

      // Principais commandos
      TButton():New(160,001,'GoUp()',oDlg,{|| oBrowse:GoUp()
},40,10,,,,.T.)
      TButton():New(170,001,'GoDown()',oDlg,{|| oBrowse:GoDown()
},40,10,,,,.T.)
      TButton():New(180,001,'GoTop()',oDlg,{|| oBrowse:GoTop()
},40,10,,,,.T.)
      TButton():New(190,001,'GoBottom()' , oDlg,{|| oBrowse:GoBottom()
},40,10,,,,.T.)
      TButton():New(160,060,'nAt (Linha selecionada)',oDlg,;
                        {|| Alert (oBrowse:nAt)},80,10,,,,.T.)
      TButton():New(170,060,'nRowCount (Nr de linhas visiveis)',oDlg,;
                        {|| Alert(oBrowse:nRowCount()) },80,10,,,,.T.)
      TButton():New(180,060, 'nLen (Numero total de linhas)', oDlg,;
                        {|| Alert(oBrowse:nLen) },80,10,,,,.T.)
      TButton():New(190,060, 'lEditCell (Edita a celula)', oDlg,;
                        {|| lEditCell(@aBrowse,oBrowse,'@!',3)
},40,10,,,,.T.)

      ACTIVATE MSDIALOG oDlg CENTERED
Return NIL
```

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.

Para visualizar a documentação completa de todos os componentes mencionados neste capítulo deve ser acesso o site DEM – Documentação Eletrônica Microsiga (dem.microsiga.com.br) conforme abaixo:

Anotações

36.1. Particularidades dos componentes visuais

36.1.1. 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()

nClrPane	Cor de fundo do painel
nClrText	Cor da fonte das letras do painel

TSAY()

nClrPane	Cor de fundo do painel
nClrText	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)

nRed	Valor de 0-255 para o elemento vermelho da paleta RGB
nGreen	Valor de 0-255 para o elemento verde da paleta RGB
nBlue	Valor de 0-255 para o elemento azul da paleta RGB
Retorno	Valor a ser definido para o atributo cor do componente

37. 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**

37.1. 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"):

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().

37.1.1. Enchoice()

☑ **Sintaxe: Enchoice(cAlias, nReg, nOpc, aCRA, cLetra, cTexto, aAcho, aPos, aCpos, nModelo, nColMens, cMensagem, cTudoOk, oWnd, lF3, lMemoria, lColumn, caTela, lNoFolder, lProperty)**

☑ **Retorno: Nil**
☑ **Parâmetros:**

cAlias	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada
nReg	Parâmetro não utilizado
nOpc	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização)
aCRA	Parâmetro não utilizado
cLetra	Parâmetro não utilizado
cTexto	Parâmetro não utilizado
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"
aPos	Vetor com coordenadas para criação da enchoice no formato {<top>, <left>, <bottom>, <right>}
aCpos	Vetor com nome dos campos que poderão ser editados
nModelo	Se for diferente de 1 desabilita execução de gatilhos estrangeiros
nColMens	Parâmetro não utilizado
cMensagem	Parâmetro não utilizado
cTudoOk	Expressão para validação da Enchoice
oWnd	Objeto (janela, painel, etc.) onde a enchoice será criada.
lF3	Indica se a enchoice esta sendo criada em uma consulta F3 para utilizar variáveis de memória
lMemoria	Indica se a enchoice utilizará variáveis de memória ou os campos da tabela na edição
lColumn	Indica se a apresentação dos campos será em forma de coluna
caTela	Nome da variável tipo "private" que a enchoice utilizará no lugar da propriedade aTela
lNoFolder	Indica se a enchoice não irá utilizar as Pastas de Cadastro (SXA)
lProperty	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    := {}
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 aTELA[0][0]
Private aGETS[0]

DbSelectArea("SX3")
DbSetOrder(1)
DbSeek(cAliasE)
```

```
Exemplo (continuação):

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*/,/*cTudoOk*/, oDlg, lF3, lMemoria, lColumn,;
        caTela, lNoFolder, lProperty)

ACTIVATE MSDIALOG oDlg CENTERED

Return
```

37.1.2. MsMGet()

☑ **Sintaxe: MsMGet():New(cAlias, nReg, nOpc, aCRA, cLetra, cTexto, aAcho, aPos, aCpos, nModelo, nColMens, cMensagem, cTudoOk, oWnd, lF3, lMemoria, lColumn, caTela, lNoFolder, lProperty)**

☑ **Retorno: oMsMGet → objeto do tipo MsMGet()**

☑ **Parâmetros:**

cAlias	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada
nReg	Parâmetro não utilizado
nOpc	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização)
aCRA	Parâmetro não utilizado
cLetra	Parâmetro não utilizado
cTexto	Parâmetro não utilizado
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"
aPos	Vetor com coordenadas para criação da enchoice no formato {<top>, <left>, <bottom>, <right>}
aCpos	Vetor com nome dos campos que poderão ser editados
nModelo	Se for diferente de 1 desabilita execução de gatilhos estrangeiros
nColMens	Parâmetro não utilizado
cMensagem	Parâmetro não utilizado
cTudoOk	Expressão para validação da Enchoice
oWnd	Objeto (janela, painel, etc.) onde a enchoice será criada.
lF3	Indica se a enchoice esta sendo criada em uma consulta F3 para utilizar variáveis de memória

lMemoria	Indica se a enchoice utilizará variáveis de memória ou os campos da tabela na edição
lColumn	Indica se a apresentação dos campos será em forma de coluna
caTela	Nome da variável tipo "private" que a enchoice utilizará no lugar da propriedade aTela
lNoFolder	Indica se a enchoice não irá utilizar as Pastas de Cadastro (SXA)
lProperty	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 aTELA[0][0]
Private aGETS[0]
```

Exemplo (continuação):

```
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,lF3,lMemoria,lColumn, caTela,;
         lNoFolder, lProperty)

oDlg:lCentered := .T.
oDlg:Activate()

Return
```

37.2. 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.

37.2.1. 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:**

nTop	Distancia entre a MsGetDB e o extremidade superior do objeto que a contém.
nLeft	Distancia entre a MsGetDB e o extremidade esquerda do objeto que a contém.
nBottom	Distancia entre a MsGetDB e o extremidade inferior do objeto que a contém.
nRight	Distancia entre a MsGetDB e o extremidade direita do objeto que a contém.
nOpc	Posição do elemento do vetor aRotina que a MsGetDB usará como referência.
cLinhaOk	Função executada para validar o contexto da linha atual do aCols.
cTudoOk	Função executada para validar o contexto geral da MsGetDB (todo aCols).
cIniCpos	Nome dos campos do tipo caracter que utilizarão incremento automático. Este parâmetro deve ser no formato "+<nome do primeiro campo>+<nome do segundo campo>+...".
lDelete	Habilita a opção de excluir linhas do aCols. Valor padrão falso.
aAlter	Vetor com os campos que poderão ser alterados.
nFreeze	Indica qual coluna não ficara congelada na exibição.
lEmpty	Habilita validação da primeira coluna do aCols para esta não poder estar vazia. Valor padrão falso.
uPar1	Parâmetro reservado.
cFieldOk	Função executada na validação do campo.
cTRB	Alias da tabela temporária.
lCondicional	Reservado
lAppend	Indica se a MsGetDB ira criar uma linha em branco automaticamente quando for inclusão.
cDelOk	Função executada para validar a exclusão de uma linha do aCols.
lDisparos	Indica se será utilizado o Dicionário de Dados para consulta padrão, inicialização padrão e gatilhos.
uPar2	Parâmetro reservado.
cSuperDel	-Função executada quando pressionada as teclas <Ctrl>+<Delete>.
oWnd	Objeto no qual a MsGetDB será criada.

☑ **Aparência:**

☑ **Variáveis private:**

aRotina	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: 11-Pesquisar 12-Visualizar 13-Incluir 14-Alterar 15-Excluir
aHeader	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.
lRefresh	Variável tipo lógica para uso reservado.

☑ **Variáveis públicas:**

nBrLin	Indica qual a linha posicionada do aCols.

☑ **Funções de validação:**

cLinhaOk	Função de validação na mudança das linhas da grid. Não pode ser definida como Static Function.
cTudoOk	Função de validação da confirmação da operação com o grid. Não pode ser definida como Static Function.

☑ **Métodos adicionais:**

ForceRefresh()	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 nI
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", "AxDeleta", 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:lCentered := .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.
```

37.2.2. 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:**

nTop	Distancia entre a MsGetDados e o extremidade superior do objeto que a contém.
nLeft	Distancia entre a MsGetDados e o extremidade esquerda do objeto que a contém.
nBottom	Distancia entre a MsGetDados e o extremidade inferior do objeto que a contém.
nRight	Distancia entre a MsGetDados e o extremidade direita do objeto que a contém.
nOpc	Posição do elemento do vetor aRotina que a MsGetDados usará como referencia.
cLinhaOk	Função executada para validar o contexto da linha atual do aCols.
cTudoOk	Função executada para validar o contexto geral da MsGetDados (todo aCols).
cIniCpos	Nome dos campos do tipo caracter que utilizarão incremento automático. Este parâmetro deve ser no formato "+<nome do primeiro campo>+<nome do segundo campo>+...".
lDelete	Habilita excluir linhas do aCols. Valor padrão falso.
aAlter	Vetor com os campos que poderão ser alterados.
uPar1	Parâmetro reservado.
lEmpty	Habilita validação da primeira coluna do aCols para esta não poder estar vazia. Valor padrão falso.
nMax	Número máximo de linhas permitidas. Valor padrão 99.
cFieldOk	Função executada na validação do campo.
cSuperDel	Função executada quando pressionada as teclas <Ctrl>+<Delete>.
uPar2	Parâmetro reservado.
cDelOk	Função executada para validar a exclusão de uma linha do aCols.
oWnd	Objeto no qual a MsGetDados será criada.

☑ **Aparência:**

☑ **Variáveis private:**

aRotina	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: 16-Pesquisar 17-Visualizar 18-Incluir 19-Alterar 20-Excluir
aHeader	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.
lRefresh	Variável tipo lógica para uso reservado.

☑ **Variáveis públicas:**

N	Indica qual a linha posicionada do aCols.

☑ **Funções de validação:**

cLinhaOk	Função de validação na mudança das linhas da grid. Não pode ser definida como Static Function.
cTudoOk	Função de validação da confirmação da operação com o grid. Não pode ser definida como Static Function.

☑ **Métodos adicionais:**

ForceRefresh()	Atualiza a MsGetDados com a tabela e posiciona na primeira linha.
Hide()	Oculta a MsGetDados.
Show()	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 nI
Local oDlg
Local oGetDados
Local nUsado := 0
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", "AxDeleta", 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,;
                      "",;
                      "" })
EndIf
DbSkip()
End

AADD(aCols,Array(nUsado+1))

For nI := 1 To nUsado
    aCols[1][nI] := CriaVar(aHeader[nI][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.
```

37.2.3. 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, cIniCpos, aAlterGDa, nFreeze, nMax, cFieldOk, cSuperDel, cDelOk, oDLG, aHeader, aCols)**

☑ **Retorno: oMsGetDados → objeto do tipo MsNewGetDados()**

☑ **Parâmetros:**

nSuperior	Distancia entre a MsNewGetDados e o extremidade superior do objeto que a contem
nEsquerda	Distancia entre a MsNewGetDados e o extremidade esquerda do objeto que a contem
nInferior	Distancia entre a MsNewGetDados e o extremidade inferior do objeto que a contem
nDireita	Distancia entre a MsNewGetDados e o extremidade direita do objeto que a contem
nOpc	Operação em execução: 2- Visualizar, 3- Incluir, 4- Alterar, 5- Excluir
cLinOk	Função executada para validar o contexto da linha atual do aCols
cTudoOk	Função executada para validar o contexto geral da MsNewGetDados (todo aCols)
cIniCpos	Nome dos campos do tipo caracter que utilizarão incremento automático.
aAlterGDa	Campos alteráveis da GetDados
nFreeze	Campos estáticos na GetDados, partindo sempre da posição inicial da getdados aonde: 3- Primeiro campo congelado 4- Primeiro e segundo campos congelados...
nMax	Número máximo de linhas permitidas. Valor padrão 99
cFieldOk	Função executada na validação do campo
cSuperDel	Função executada quando pressionada as teclas <Ctrl>+<Delete>
cDelOk	Função executada para validar a exclusão de uma linha do aCols
oDLG	Objeto no qual a MsNewGetDados será criada
aHeader	Array a ser tratado internamente na MsNewGetDados como aHeader
aCols	Array a ser tratado internamente na MsNewGetDados como aCols

☑ **Aparência:**

☑ **Variáveis private:**

aRotina	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
aHeader	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.
lRefresh	Variável tipo lógica para uso reservado.

☑ **Variáveis públicas:**

N	Indica qual a linha posicionada do aCols.

☑ **Funções de validação:**

cLinhaOk	Função de validação na mudança das linhas da grid. Não pode ser definida como Static Function.
cTudoOk	Função de validação da confirmação da operação com o grid. Não pode ser definida como Static Function.

☑ **Métodos adicionais:**

ForceRefresh()	Atualiza a MsNewGetDados com a tabela e posiciona na primeira linha.
Hide()	Oculta a MsNewGetDados.
Show()	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  := {}
```

Exemplo (continuação):

```
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 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
```

```
Exemplo (continuação):

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,lF3, lMemoria,lColumn,caTela,lNoFolder,;
            lProperty)

      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:lCentered := .T.
oDlg:Activate()
Return
```

37.2.3.1. 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:**

lUseDefaultColors	Atributo que deverá ser definido como .F. para que as alterações nas cores sejam permitidas.

☑ **Métodos adicionais:**

SetBlkBackColor	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    | MRBWGTCL   | 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
```

```
Exemplo (continuação):

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
```

```
Exemplo (continuação):

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,lF3, lMemoria,lColumn,caTela,lNoFolder,lProperty)

      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({|| GETDCLR(oGetD:aCols,oGetD:nAt,aHeader)})

oDlg:bInit  := {|| EnchoiceBar(oDlg, {||oDlg:End()}, {||oDlg:End()},,aButtons)}
oDlg:lCentered := .T.
oDlg:Activate()

Return
```

```
Exemplo (continuação):

// 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
```

Anotações

37.3. 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.

37.3.1. EnchoiceBar()

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:**

```
EnchoiceBar( oDlg, bOk, bCancel, lMsgDel, aButtons, nRecno, cAlias)
```

☑ **Parâmetros:**

oDlg	Dialog onde irá criar a barra de botões
bOk	Bloco de código a ser executado no botão Ok
bCancel	Bloco de código a ser executado no botão Cancelar
lMsgDel	Exibe dialog para confirmar a exclusão
aButtons	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
nRecno	Registro a ser posicionado após a execução do botão Ok.
cAlias	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,{||lOk:=.T.,oDlg:End()},{||oDlg:End()},,@aButtons))

Return
```

Anotações

37.3.2. 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:**

oWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
nBtnWidth	Numérico, opcional. Largura do botão contido na barra
nBtnHeight	Numérico, opcional. Altura do botão contido na barra
l3D	Lógico, opcional. Define tipo da barra
cMode	Não utilizado.
oCursor	Objeto, opcional. Define Cursor ao posicionar o mouse sobre a barra.
cResource	Caracter, opcional. Imagem do recurso a ser inserido como fundo da barra.
lNoAutoAdjust	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:lCentered := .T.
oDlg:Activate()

Return
```

37.3.3. 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:**

DEFINE BUTTONBAR oBar SIZE nWidth, nHeight 3D MODE OF oDlg CURSOR

☑ **Retorno: ().**

☑ **Parâmetros:**

oBar	Objeto do tipo TBar() que será criado com a utilização da sintaxe ButtonBar().
nWidth	Numérico, opcional. Largura do botão contido na barra.
nHeight	Numérico, opcional. Altura do botão contido na barra.
3D	Se definido habilita a visualização em 3D da barra de botões.
oDlg	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
MODE	Define a forma de orientação do objeto ButtonBar utilizando os seguintes termos pré-definidos: TOP, BOTTOM, FLOAT
CURSOR	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:**

cBitMap	Nome da imagem disponível na aplicação.
oBar	Objeto do tipo TBar() no qual o botão será adicionado.
cAcao	Função ou lista de expressões que determina a ação que será realizada pelo botão.
cTexto	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
```

Exemplo (continuação):

```
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
```

Anotações

37.3.4. 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:**

cBitMap	Nome da imagem pré-definida existente na aplicação ERP que será vinculada ao botão.
bAcao	Bloco de código que define a ação que será executada com a utilização do botão.
cTexto	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"})
```

38. Outras aplicações da interface visual do ADVPL

38.1. MaWndBrowse()

☑ **Descrição:** Browse que permite a visualização de registros para arquivos / tabelas que não possuem estrutura definida no Dicionário de Dados do sistema.

☑ **Sintaxe:** MaWndBrowse (nLin1, nCol1, nLin2, nCol2, cTitle, cAlias, aCampos, aRotina, cFunLeg, cTopFun, cBotFun, lCentered, aResource, nMod, aPesqui, cSeek, lDic, lSavOrd)

☑ **Parâmetros:**

nLin1	Linha inicial do browse
nCol1	Coluna inicial do browse
nLin2	Linha final do browse
nCol2	Coluna final do browse
cTitle	Título do browse (obrigatório)
cAlias	Alias da tabela corrente podendo ser um temporário
aCampos	Se lDic=.T. utilizará o SX3, do contrário o aCampos informado
aRotina	Idêntico ao aRotina para mBrowse
cFunLeg	Função que deverá retornar um valor lógico e com isso será atribuído semafóro na primeira coluna do browse
cTopFun	Mostrar os registros com a chave de
cBotFun	Mostrar os registros com a chave ate
lCentered	Valor verdadeiro centraliza
aResource	aAdd(aResource,{"IMAGEM","Texto significativo"})
nModelo	Posição do Menu: 1- Horizontal (superior) 2- Vertical (esquerda) 3- Horizontal (inferior)
aPesqui	aAdd(aPesqui{"Título",nOrdem}), se não passado será utilizado o AxPesqui
cSeek	Chave principal para a busca, exemplo: xFilial("???")
lDic	Parâmetro em conjunto com aCampos
lSavOrd	Estabelecer a ordem após pesquisas.

☑ **Estrutura do array aCampos**

aAdd(aCampo,{X3_CAMPO,X3_PICTURE,X3_TITULO,X3_TAMANHO})

☑ **Aparência:**

Exemplo: MaWndBrowse para arquivo temporário

```
#include "protheus.ch"

/*/
+-----------------------------------------------------------------------
| Função    | WndBrwTRB     | Autor | Arnaldo R. Junior | Data |      |
+-----------------------------------------------------------------------
| Descrição | Demonstra a utilização da MaWndBrowse com Temporário    |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                              |
+-----------------------------------------------------------------------
/*/

User Function WndBrwTRB()

// Variáveis para o Arquivo Temporario
Local cChave      := ""
Local cArqTrb     := ""
Local aStruTRB  :=  {}

// Variáveis para o MaWndBrowse
Local cTitulo   := "Cadastro Temporario"// Título obrigatório
Local cAlias    := "TRB" // Alias da tabela corrente podendo ser TRB
Local cFunLeg   := "" // Função que deverá retornar um valor lógico e com isso
será atribuído semaféro na primeira coluna do browse
Local cTopFun   := "" // Mostrar os registros com a chave de
Local cBotFun   := "" // Mostrar os registros com a chave ate
Local lCentered := .T. // Valor verdadeiro centraliza
Local aResource := {} // aAdd(aResource,{"IMAGEM","Texto significativo"})
Local nModelo   := 1 // 1- Menu do aRotina
Local aPesqui   := {} // aAdd(aPesqui{"Título",nOrdem}), se não passado será
utilizado o AxPesqui
Local cSeek     := "" // Chave principal para a busca, exemplo: xFilial("???")
Local lDic      := .F. // Parâmetro em conjunto com aCampos
Local lSavOrd   := .T. // Estabelecer a ordem após pesquisas.
```

```
// Variaveis para a MsAdvSize
Local lEnchBar  := .F. // Se a janela de diálogo possuirá enchoicebar (.T.)
Local lPadrao   := .F. // Se a janela deve respeitar as medidas padrões do
//Protheus (.T.) ou usar o máximo disponível (.F.)
Local nMinY     := 400 // Altura mínima da janela

Local aSize       := MsAdvSize(lEnchBar, lPadrao, nMinY)

Private cCadastro := " "
Private aCampos     := {} // Se lDic=.T. utilizará o SX3, do contrário o aCampos
//informado -> aAdd(aCampo,{X3_CAMPO,X3_PICTURE,X3_TITULO,X3_TAMANHO})
Private aRotina   := {} // Idêntico ao aRotina para mBrowse

aAdd(aRotina,{"Pesquisar" ,"AxPesqui",0,1})
aAdd(aRotina,{"Visualizar","U_TVisual",0,2})
aAdd(aRotina,{"Incluir"   ,"U_TInclui",0,3})
aAdd(aRotina,{"Alterar"   ,"U_TAltera",0,4})
aAdd(aRotina,{"Excluir"   ,"U_TExclui",0,5})

// Estrutura do Arquivo: Nome do campo / tipo, tamanho, decimais (SX3 para
//temporário)
AADD(aStruTRB,{"TRB_FILIAL"        ,"C",02,0}) // Nome_Campo , Tipo_Campo,
//Tamanho, Decimal
AADD(aStruTRB,{"TRB_ID"            ,"C",14,0})
AADD(aStruTRB,{"TRB_NOME"          ,"C",20,0})
AADD(aStruTRB,{"TRB_IDADE"         ,"N",03,0})
AADD(aStruTRB,{"TRB_STATUS"        ,"C",01,0})

// aCampos padrão para a MaWndBrowse
//AADD(aCampos,{<Nome_Campo>,<Picture>,<Titulo>,<Tamanho>})

// aCampos melhorado para a WndBrwTRB
//AADD(aCampos,{<Nome_Campo>,<Picture>,<Titulo>,<Tamanho>,<Tipo>,<cWhen>,<lObrig
//atorio>})
// Nota: lObrigatorio deve ser sempre a ultima informacao do aCampos

AADD(aCampos,{"TRB_FILIAL"    ,"@!" ,"Filial"         ,02,"C",".F.",.T.})
AADD(aCampos,{"TRB_ID"        ,"@!" ,"Matricula"      ,14,"C",".F.",.T.})
AADD(aCampos,{"TRB_NOME"      ,"@!" ,"Nome"                ,20,"C",".T.",.F.})
AADD(aCampos,{"TRB_IDADE"     ,"@E 999","Idade"       ,03,"N",".T.",.F.})
AADD(aCampos,{"TRB_STATUS"    ,"@!" ,"Status"         ,01,"C",".T.",.T.})

If ( Select( "TRB" ) <> 0 )
      dbSelectArea ( "TRB" )
      dbCloseArea ()
Endif

//ÚÄÄÄ
```

```
dbSelectArea( "TRB" )
MaWndBrowse(aSize[7],aSize[2],aSize[6],aSize[5],cTitulo,cAlias,aCampos,aRotina,,
cTopFun,cBotFun,lCentered,,nModelo,,cSeek,lDic,lSavOrd)

If ( Select( "TRB" ) <> 0 )
      dbSelectArea ( "TRB" )
      dbCloseArea ()
Endif

Return
```

38.1.1. Enchoice para Arquivos Temporários

Para aproveitar inteiramente o recurso da MaWndBrowse utilizada com arquivos temporários, é necessário implementar uma função do tipo Enchoice para realizar as operações de Visualizar, Incluir, Alterar e Excluir os registros deste arquivo / tabela.

Com foco nesta necessidade, foi desenvolvida para o curso de ADVPL a função TEnchoice.

☑ **Descrição:** Função para manipulação de registros de um arquivo temporário simular a Enchoice do ambiente Protheus.

☑ **Sintaxe:** TEnchoice(oDlg, aCampos, nLeftE, nTopE, nHeightE, nWidthE, lEnchBar)

☑ **Parâmetros:**

oDlg	Objeto Dialog no qual a TEnchoice será vinculada
aCampos	Estrutura de campos para exibição da tela da TEnchoice
nLeftE	Coordenada horizontal em pixels.
nTopE	Coordenada vertical em pixels.
nHeightE	Altura em pixels
nWidthE	Largura em pixels.
lEnchBar	Se a enchoice será montada em conjunto com uma barra de botões do tipo Enchoicebar()

☑ **Pré-requisitos da função chamadora:**

Para correta utilização da função TEnchoice são necessários os seguintes tratamentos / definições na função chamadora:

☑ **Tratamento 01: Montagem do array aCampos**

O array aCampos utilizado pela TEnchoice possui basicamente o mesmo formato do array utilizado pela função Modelo2().

"<Variavel>"	Nome da variável Private que está vinculada ao campo digitável.
{nLinha,nColuna}	Posição em pixels do campo na Enchoice.
"<Titulo>"	Título do campo.
"<Picture>"	Picture de formatação do campo.
"<Validacao>"	Validação do campo digitável.
"<F3>"	Consulta F3 vinculada ao campo.

"<lWhen>"	Se o campo está editável.
<Tamanho>	Tamanho do campo para visualização.

```
AADD(aCposEnch,   {"<Variavel>"     ,{nLinha,nColuna} ,"<Titulo>",;
                   "<Picture>", "<Validacao>", "<F3>", "<lWhen>",;
                   <Tamanho>})
```

☑ **Tratamento 02: Definição das dimensões da TEnchoice**

Os parâmetros de dimensão da TEnchoice não precisam se informados, pois ela irá se ajustar ao tamanho do objeto Dialog ao qual for vinculado, caso o objetivo seja que a TEnchoice ocupe toda a área disponível do objeto.

É importante especificar se o Dialog irá conter uma EnchoiceBar() para que a TEnchoice() seja posicionada adequadamente no objeto.

☑ **Aparência:**

☑ **Detalhamento da função TEnchoice:**

```
/*/
+-----------------------------------------------------------------------
| Função    | TEnchoice     | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | Enchoice para arquivos temporarios                        |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

Static Function TEnchoice(oDlg, aCampos, nLeftE, nTopE, nHeightE, nWidthE,;
lEnchBar)

Local aSays       := {}
Local aGets       := {}
Local cCaption    := ""
Local cPict       := ""
Local cValid      := ""
Local cF3         := ""
Local cWhen       := ""
Local cBlKSay     := ""
Local cBlkGet     := ""
Local cBlKVld     := ""
Local cBlKWhen    := ""
Local nLeft       := 0
Local nTop        := 0
Local nI          := 0

Default lEnchBar  := .F.
Default     nLeftE            := IIF(lEnchBar,(oDlg:nLeft)+16,2)
Default     nTopE       := 2
Default     nHeightE    := (oDlg:nHeight)-135
Default nWidthE         := (oDlg:nWidth)-102
// Padrao Modelo2(), com lWhen -> cWhen
// AADD(aC,{"<Variavel>"      ,{nLinha,nColuna}
//,"<Titulo>","<Picture>","<Validacao>","<F3>","<lWhen>",<Tamanho>})

If Len(aCampos) > 0

    oScroll := TScrollBox():New(oDlg, nLeftE, nTopE, nHeightE , nWidthE, .T.,;
.T., .T.)

      For nI:=1 to Len(aCampos)

            If Len(aCampos[nI])==8

                  cCampo      := aCampos[nI,1]
                  nLeft := aCampos[nI,2,1]-13
                  nTop  := aCampos[nI,2,2]
                  cCaption:= Iif(Empty(aCampos[nI,3]), " "  ,aCampos[nI,3])
                  cPict := Iif(Empty(aCampos[nI,4]), Nil    ,aCampos[nI,4])
                  cValid      := Iif(Empty(aCampos[nI,5]), ".t."  ,aCampos[nI,5])
                  cF3         := Iif(Empty(aCampos[nI,6]), NIL    ,aCampos[nI,6])
                  cWhen := Iif(Empty(aCampos[nI,7]), ".T."  ,aCampos[nI,7])


                  nWidthG     := Iif(Empty(aCampos[nI,8]), 100,;
                                 IiF(aCampos[nI,8]*3.5 > 100,100,nil))
```

```
                    cBlKSay := "{|| OemToAnsi('"+cCaption+"')}"
                    cBlkGet := "{ | u | If( PCount() == 0,;
                               "+cCampo+","+cCampo+":= u ) }"
                    cBlKVld := "{|| "+cValid+"}"
                    cBlKWhen := "{|| "+cWhen+"}"

                    AADD(aSays,Array(1))
                    aSays[nI] := TSay():New(nLeft+1, nTop, &(cBlkSay), oScroll,,,;
                                .F., .F., .F., .T.,,, 50, 8, .F., .F., .F., .F.,;
                                .F. )
                    AADD(aGets,Array(1))

                    aGets[nI] := TGet():New( nLeft, nTop+50, &(cBlKGet), oScroll,;
                                 nWidthG, 11, cPict, &(cBlkVld),,,, .F.,, .T.,,;
                                 .F., &(cBlkWhen), .F., .F.,, .F., .F. , cF3,;
                                 (cCampo))

            EndIf
      Next
Endif

Return
```

Anotações

Exemplo: Função TVisual (MaWndBrowse com TEnchoice)

```
/*/
+-----------------------------------------------------------------------
| Função    | TVisual       | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | Enchoice para arquivos temporarios                        |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

USER FUNCTION TVisual(cAlias,nReg,nOpc)

LOCAL aCposEnch := {}
LOCAL nLinha      := 15
LOCAL nColuna     := 10
LOCAL nOpcE       := aRotina[nOpc][4] // Opcao de verdade
LOCAL bOk         := {||oDlg:End()}
LOCAL bCancel     := {||oDlg:End()}
LOCAL nX

// Padrao Modelo2(), com lWhen -> cWhen
// AADD(aCposEnch,{"<Variavel>"     ,{nLinha,nColuna}
//,"<Titulo>","<Picture>","<Validacao>","<F3>","<lWhen>",<Tamanho>})
// aCampos,{"TRB_FILIAL"            ,"@!" ,"Filial"         ,02}

For nX := 1 to Len(aCampos)

      If nX > 1
            nLinha  := IIF(nX%2 > 0 ,nLinha  := nLinha +015,nLinha) // Impar
            nColuna := IIF(nX%2 == 0,nColuna := nColuna+170,10)// Par
      Endif

      AADD(aCposEnch,{"_"+aCampos[nX][1],{nLinha,nColuna} ,;
                        aCampos[nX][3],aCampos[nX][2],"AllwaysTrue()",;
                        "",".F.",aCampos[nX][4]})

      SetPrvt("_"+aCampos[nX][1])

      &("_"+aCampos[nX][1]) := (cAlias)->&(aCampos[nX][1])

Next nX

oDlg := TDialog():New(000,000,400,650,cCadastro,,,,,,,,,.T.)

TEnchoice(oDlg, aCposEnch,,,,,.T.)

oDlg:bInit := {|| EnchoiceBar(oDlg, bOk, bCancel,.F.,{},nReg,cAlias)}
oDlg:lCentered := .T.
oDlg:Activate()

RETURN
```

Exemplo: Função TInclui (MaWndBrowse com TEnchoice)

```
/*/
+-----------------------------------------------------------------------
| Função    | TInclui       | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | Enchoice para arquivos temporarios                        |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

USER FUNCTION TInclui(cAlias,nReg,nOpc)

LOCAL aCposEnch := {}
LOCAL nLinha      := 15
LOCAL nColuna     := 10
LOCAL nOpcE       := aRotina[nOpc][4] // Opcao de verdade
LOCAL bOk
LOCAL bCancel     := {||oDlg:End()}
LOCAL aArea       := GetArea()
LOCAL nX

// Padrao Modelo2(), com lWhen -> cWhen
// AADD(aCposEnch,{"<Variavel>"     ,{nLinha,nColuna}
//,"<Titulo>","<Picture>","<Validacao>","<F3>","<lWhen>",<Tamanho>})
// aCampos,{"TRB_FILIAL"            ,"@!" ,"Filial"         ,02}

For nX := 1 to Len(aCampos)

      If nX > 1
            nLinha  := IIF(nX%2 > 0 ,nLinha  := nLinha +015,nLinha) // Impar
            nColuna := IIF(nX%2 == 0,nColuna := nColuna+170,10)// Par
      Endif

      AADD(aCposEnch,{"_"+aCampos[nX][1],{nLinha,nColuna} ,;
                        aCampos[nX][3],aCampos[nX][2],"AllwaysTrue()",;
                        "",".T.",aCampos[nX][4]})

      SetPrvt("_"+aCampos[nX][1])

      Do Case
            Case aCampos[nX][5] == "C"
                  &("_"+aCampos[nX][1]) := Space(aCampos[nX][4])
            Case aCampos[nX][5] == "N"
                  &("_"+aCampos[nX][1]) := 0
            Case aCampos[nX][5] == "D"
                  &("_"+aCampos[nX][1]) := CTOD("")
            Case aCampos[nX][5] == "L"
                  &("_"+aCampos[nX][1]) := .F.
            Case aCampos[nX][5] == "M"
                  &("_"+aCampos[nX][1]) := Space(aCampos[nX][4])
      EndCase

Next nX

oDlg := TDialog():New(000,000,400,650,cCadastro,,,,,,,,,.T.)
```

```
TEnchoice(oDlg, aCposEnch,,,,,.T.)

bOk := {|| IIF( U_TValid(cAlias,nReg,nOpcE,aCampos),;
                        (U_TGravar(cAlias,nReg,nOpcE,aCampos),oDlg:End()),)}

oDlg:bInit := {|| EnchoiceBar(oDlg, bOk, bCancel,.F.,{},nReg,cAlias)}
oDlg:lCentered := .T.
oDlg:Activate()

RETURN
```

Exemplo: Função TAltera (MaWndBrowse com TEnchoice)

```
/*/
+-----------------------------------------------------------------------
| Função    | TAltera       | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | Enchoice para arquivos temporarios                        |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

USER FUNCTION TAltera(cAlias,nReg,nOpc)

LOCAL aCposEnch := {}
LOCAL nLinha      := 15
LOCAL nColuna     := 10
LOCAL nOpcE       := aRotina[nOpc][4] // Opcao de verdade
LOCAL bOk
LOCAL bCancel     := {||oDlg:End()}
LOCAL aArea       := GetArea()
LOCAL nX

// Padrao Modelo2(), com lWhen -> cWhen
// AADD(aCposEnch,{"<Variavel>"     ,{nLinha,nColuna}
//,"<Titulo>","<Picture>","<Validacao>","<F3>","<lWhen>",<Tamanho>})
// aCampos,{"TRB_FILIAL"            ,"@!" ,"Filial"         ,02}

For nX := 1 to Len(aCampos)

      If nX > 1
            nLinha  := IIF(nX%2 > 0 ,nLinha  := nLinha +015,nLinha) // Impar
            nColuna := IIF(nX%2 == 0,nColuna := nColuna+170,10)// Par
      Endif

      AADD(aCposEnch,{"_"+aCampos[nX][1],{nLinha,nColuna} ,;
                        aCampos[nX][3],aCampos[nX][2],"AllwaysTrue()",;
                        "",aCampos[nX][6],aCampos[nX][4]})

      SetPrvt("_"+aCampos[nX][1])

      &("_"+aCampos[nX][1]) := (cAlias)->&(aCampos[nX][1])

Next nX

oDlg := TDialog():New(000,000,400,650,cCadastro,,,,,,,,.T.)
```

```
TEnchoice(oDlg, aCposEnch,,,,,.T.)

bOk := {|| IIF( U_TValid(cAlias,nReg,nOpcE,aCampos),;
                        (U_TGravar(cAlias,nReg,nOpcE,aCampos),oDlg:End()),)}

oDlg:bInit := {|| EnchoiceBar(oDlg, bOk, bCancel,.F.,{},nReg,cAlias)}
oDlg:lCentered := .T.
oDlg:Activate()

RETURN
```

Exemplo: Função TExclui (MaWndBrowse com TEnchoice)

```
/*/
+-----------------------------------------------------------------------
| Função    | TExclui       | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | Enchoice para arquivos temporarios                        |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

USER FUNCTION TExclui(cAlias, nReg, nOpc)

LOCAL aCposEnch := {}
LOCAL nLinha      := 15
LOCAL nColuna     := 10
LOCAL nOpcE       := aRotina[nOpc][4] // Opcao de verdade
LOCAL bOk
LOCAL bCancel     := {||oDlg:End()}
LOCAL nX

// Padrao Modelo2(), com lWhen -> cWhen
// AADD(aCposEnch,{"<Variavel>"     ,{nLinha,nColuna}
//,"<Titulo>","<Picture>","<Validacao>","<F3>","<lWhen>",<Tamanho>})
// aCampos,{"TRB_FILIAL"            ,"@!" ,"Filial"         ,02}

For nX := 1 to Len(aCampos)

      If nX > 1
            nLinha  := IIF(nX%2 > 0 ,nLinha  := nLinha +015,nLinha) // Impar
            nColuna := IIF(nX%2 == 0,nColuna := nColuna+170,10)// Par
      Endif

      AADD(aCposEnch,{"_"+aCampos[nX][1],{nLinha,nColuna} ,;
                        aCampos[nX][3],aCampos[nX][2],"AllwaysTrue()",;
                        "",".F.",aCampos[nX][4]})

      SetPrvt("_"+aCampos[nX][1])

      &("_"+aCampos[nX][1]) := (cAlias)->&(aCampos[nX][1])

Next nX

oDlg := TDialog():New(000,000,400,650,cCadastro,,,,,,,,,.T.)
```

```
TEnchoice(oDlg, aCposEnch,,,,,.T.)

bOk := {|| IIF( U_TValid(cAlias,nReg,nOpcE,aCampos),;
                        (U_TGravar(cAlias,nReg,nOpcE,aCampos),oDlg:End()),)}

oDlg:bInit := {|| EnchoiceBar(oDlg, bOk, bCancel,.F.,{},nReg,cAlias)}
oDlg:lCentered := .T.
oDlg:Activate()

RETURN
```

Exemplo: Função TValid (MaWndBrowse com TEnchoice)

```
/*/
+-----------------------------------------------------------------------
| Função    | TValid        | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | Enchoice para arquivos temporarios                        |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

USER FUNCTION TValid(cAlias,nReg,nOpc,aCampos)
LOCAL lRet
LOCAL nX
LOCAL nPosObrig := Len(aCampos[1])

For nX := 1 to Len(aCampos)
      IF aCampos[nX,nPosObrig] == .T.
            IF !(lRet := !Empty(&("_"+aCampos[nX,1])))
                  Help("TEnchoice",1,"HELP","OBRIGATORIO","Existem campos
obrigatorios nao preenchidos",1,0)
                  RETURN lRet // EXIT
            ENDIF
      ENDIF
Next nX

IF nOpc == 3
      IF !(lRet := !((cAlias)->(dbSeek(_TRB_FILIAL+_TRB_ID))))
            Help("TEnchoice",1,"HELP","INCLUSAO","Ja existe um registro com esta
chave",1,0)
      ENDIF
ELSE
      IF !(lRet := (cAlias)->(dbSeek(_TRB_FILIAL+_TRB_ID)))
            Help("TEnchoice",1,"HELP","ALTERACAO","Nao existe um registro com
esta chave",1,0)
      ENDIF
ENDIF

RETURN lRet
```

Exemplo: Função TGravar (MaWndBrowse com TEnchoice)

```
/*/
+-----------------------------------------------------------------------
| Função    | TGravar       | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | Enchoice para arquivos temporarios                        |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

USER FUNCTION TGravar(cAlias,nReg,nOpc,aCampos)
LOCAL nX

RecLock(cAlias,nOpc==3)
      IF nOpc == 5
            DbDelete()
      ELSE
            For nX := 1 to Len(aCampos)
                  (cAlias)->&(aCampos[nX][1]) := &("_"+aCampos[nX][1])
            Next nX
      ENDIF
Msunlock()

RETURN
```

Anotações

38.2. DbTree()

☑ **Descrição:** Classe Advpl que permite a criação de um objeto visual do tipo Tree.

☑ **Propriedades:**

bChange	Armazena o code-block executado quando da mudança de foco entre elementos do Tree atual.

☑ **Métodos principais:**

New	Contrutor da Classe DbTree. Retorna uma nova instância do Objeto da Classe DbTree.
AddTree	Através do método AddTree, é possível acrescentar um 'nó' na árvore atual. Uma vez acrescentado o nó, as próximas inclusões de itens na parvore serão realizadas neste nó, até que o mesmo seja fechado através do método EndTree.
AddTreeItem	Através dele, podemos acrescentar itens na árvore atual ou último nó acrescentado em aberto.
EndTree	Deve ser utilizado após todos os elementos e itens serem acrescentados a um nó da árvore, aberto com o método AddTree().

☑ **Aparência:**

Método: New()

☑ **Sintaxe:** DbTree():New ([nTop] , [nLeft] , [nBottom] , [nRight] , [oWnd] , [bchange] , [bRClick] , [lCargo] , [lDisable])

☑ **Parâmetros:**

nTop	Coordenada vertical superior do Objeto.
nLeft	Coordenada horizontal esquerda do Objeto.
nBottom	Coordenada vertical inferior do Objeto.
nRight	Coordenada horizontal direita do Objeto.
oWnd	Janela pai do Objeto Tree
bchange	Code-Block contendo a ação a ser edxecutada na mudança de foco enrte os elementos da árvore.
bRClick	Code-Block a ser executado quando pressionado o botão direito do Mouse sobre um elemento da árvoew.
lCargo	Se .T., indica que os elementos do Tree utilizarão a propriedade CARGO, capaz de armazenar uma string identificadora, fornecida na montagem para cada elemento e item da árvore.
lDisable	Se .T., cria o objeto do Tree desabilitado, não permitindo foco e navegação no mesmo qneuando ele não seja habilitado.

☑ **Retorno:**

Objeto	Retorna uma nova instância do Objeto da Classe DbTree.

Método: AddTree()

☑ **Sintaxe:** oObj:AddTree (< cLabel > , [lPar02] , [cResOpen] , [cResClose] , [cBMPOpen] , [cBMPClose] , [cCargo])

☑ **Parâmetros:**

cLabel	Título do "nó" da árvore a ser acrescentado.
lPar02	(reservado)
cResOpen	Resource do RPO a ser utilizado quando o nó estiver aberto.
cResClose	Resource do RPO (bitmap) a ser utilizado quando o nó estiver fechado.
cBMPOpen	Path + Imagem bitmap a ser utilizada quando o n[o estiver aberto. A especificação de imagens a partir dos parametros cBMPOpen e cBMPClose apenas será considerada caso os parâmetros cResOpen e cResClose não forem especificados.
cBMPClose	Path + Imagem bitmap a ser utilizada quando o nó estiver fechado. A especificação de imagens a partir dos parametros cBMPOpen e cBMPClose apenas será considerada caso os

	parâmetros cResOpen e cResClose não forem especificados.
cCargo	String com valor único, fornecido para identificar o nó atual. Pode ser recuperado posteriormente durante a navegação.

☑ **Retorno:**

Nenhum	-

Método: AddTreeItem()

☑ **Sintaxe:** oObj:AddTreeItem (<cLabel> , [cResource] , [cBitMap] , [cCargo])

☑ **Parâmetros:**

cLabel	Título do item da árvore.
cResource	Resource (BMP) do repositório, a ser mostrado à esquerda do título do item.
cBitMap	Arquivo de imagem (bitmap) a ser mostrado à esquerda do título do item. Este parâm,etro apenas serã considerado caso o parâmetro cResource não for informado.
cCargo	através dele, podemos especificar uma string de identificação para este item na árvore.

☑ **Retorno:**

Nenhum	-

Método: EndTree()

☑ **Sintaxe:** oObj:EndTree ( )

☑ **Parâmetros:**

Nenhum	

☑ **Retorno:**

Nenhum	

Exemplo:

```
#include "Protheus.ch"

/*/
+-----------------------------------------------------------------------
| Função    | TDBTree       | Autor | MICROSIGA         | Data |       |
+-----------------------------------------------------------------------
| Descrição | Exemplo de utilização da função DbTree                    |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

User Function TDBTree()
Local cBmp1 := "PMSEDT3"
Local cBmp2 := "PMSDOC"
Private cCadastro := "Meu Computador"
Private oDlg
Private oDBTree

DEFINE MSDIALOG oDlg TITLE cCadastro FROM 0,0 TO 240,500 PIXEL

   oDBTree := dbTree():New(10,10,95,240,oDlg,{|| U_Proc(oDBTree:GetCargo())},;
                           ,.T.)

            oDBTree:AddTree("Pentium 4"+Space(24),.T.,cBmp1,cBmp1,,,"1.0")

                  oDBTree:AddTreeItem("Gabinete",cBmp2,,"1.1")
                  oDBTree:AddTreeItem("Monitor" ,cBmp2,,"1.2")
                  oDBTree:AddTreeItem("Teclado" ,cBmp2,,"1.3")
                  oDBTree:AddTreeItem("Mouse",cBmp2,,"1.4")
                  oDBTree:AddTreeItem("Som",cBmp2,,"1.5")
            oDBTree:AddTree("Placa Mãe",.T.,cBmp1,cBmp1,,,"2.0")
                  oDBTree:AddTreeItem("Processador",cBmp2,,"2.1")
                  oDBTree:AddTreeItem("Memória" ,cBmp2,,"2.2")
                  oDBTree:AddTreeItem("Vídeo",cBmp2,,"2.3")
                  oDBTree:AddTreeItem("Fonte",cBmp2,,"2.4")

            oDBTree:EndTree()

      oDBTree:EndTree()

DEFINE SBUTTON FROM 107,213 TYPE 1 ACTION oDlg:End() ENABLE OF oDlg
ACTIVATE MSDIALOG oDlg CENTER

Return
```

```
User Function Proc(cCargo)
Local cRet := ""

If     cCargo == "1.1"
MsgInfo("Gabinete Torre com 4 baias - Preto, com unidade de leitura e”+;
“gravação de CD/DVD",cCadastro)
Elseif cCargo == "1.2"
      MsgInfo("Monitor LCD 17' - LG",cCadastro)
Elseif cCargo == "1.3"
      MsgInfo("Teclado multimídia com funções de internet e e-mail",cCadastro)
Elseif cCargo == "1.4"
      MsgInfo("Mouse Optico sem fio",cCadastro)
Elseif cCargo == "1.5"
      MsgInfo("2 Caixas de Som - 50W RMS Cada",cCadastro)
Elseif cCargo == "2.1"
      MsgInfo("Processador Pentium 4 - 3.8 Ghz",cCadastro)
Elseif cCargo == "2.2"
      MsgInfo("1 Pente de Memória de 1Gb - DDR 500",cCadastro)
Elseif cCargo == "2.3"
      MsgInfo("Placa de Vídeo GeoForce 5000 com 256Mb",cCadastro)
Elseif cCargo == "2.4"
      MsgInfo("Fonte de Alimentação de 500W",cCadastro)
Endif

Return
```

Anotações

38.3. MsSelect()

☑ **Descrição:** A classe MsSelect cria um objeto browse (ou grid), com a primeira coluna sendo do tipo marcação.

☑ **Propriedades:**

oWnd	Objeto pai da MsSelect (MSDIALOG, MSWINDOW, MSPANEL)
oBrowse	Objeto browse (ou grid) da MsSelect, herdado da classe TCBROWSE

☑ **Métodos principais:**

New	Contrutor da Classe MsSelect. Retorna uma nova instância do Objeto da Classe MsSelect.

☑ **Aparência:**

Anotações

Método: New()

☑ **Sintaxe:** MsSelect():NEW (< cAlias > , [cCampo] , [cCpo] , [aCampos] , [lInv] , [cMar] , [aCord] , [cTopFun] , [cBotFun] , [oWnd] , [reserved] , [aColors])

☑ **Parâmetros:**

cAlias	Tabela que será utilizada pela MsSelect
cCampo	Campo que será utilizado na coluna de marcação
cCpo	Campo da tabela informada pelo parametro cAlias ou função que será executada na apresentação da coluna que indica se a linha da MsSelect esta habilitada ou não
aCampos	Vetor com informações dos campos para criação das colunas da MsSelect. Se não for informado, a MsSelect irá criar as colunas baseado no Dicionário de Campos (SX3) da tabela informada no parametro cAlias. Formato: 1 - campo ou blocó de código com conteúdo que será apresentado na coluna pela MsSelect 2 - não utilizado 3 - cabeçalho 4 - mácara de apresentação (picture)
lInv	Indica se MsSelect irá utilizar marcação invertida
cMar	Marca que será utilizada pela MsSelect para controle do campo informado pelo parametro cCampo. Para utilizar o parametro cMar, utilize a função GetMark para retornar a próxima marca.
aCord	Coordenadas para criação da MsSelect. Formato: 1 - Linha inicial 2 - Coluna inicial 3 - Linha final 4 - Coluna final
cTopFun	Função que retornará o conteúdo inicial que a MsSelect utilizará para apresentar a primeira linha da tabela, como um range, junto com o parametro cBotFun. O conteúdo retornado será utilizado para fazer o posicionamento da tabela informada pelo parametro cAlias, baseado na chave de índice posicionada para a mesma.
cBotFun	Função que retornará o conteúdo final que a MsSelect utilizará para apresentar a última linha da tabela, como um range, junto com o parametro cTopFun. O conteúdo retornado será utilizado para o posicionamento final da tabela informada pelo parametro cAlias, baseado na chave de índice posicionada para a mesma
oWnd	Objeto pai da MsSelect (MSDIALOG, MSWINDOW, MSPANEL, etc.)
reserved	Parametro reservado
aColors	Vetor com regras para a apresentação da coluna de legenda. Formato: 1 - Expressão ADVPL com retorno lógico que indica se a cor será utilizada pela coluna 2 - cor da coluna:

	BR_AMARELO BR_AZUL BR_BRANCO BR_CINZA BR_LARANJA BR_MARROM BR_PINK BR_PRETO BR_VERDE BR_VERMELHO

☑ **Retorno:**

Objeto	Retorna uma nova instância do Objeto da Classe MsSelect.

Exemplo:

```
DEFINE MSDIALOG oDlg1 TITLE "Selecao para Transferencia" From 9,0 To 32,80;
OF oMainWnd
      @1.4,.8 Say "Valor Total:"
      @1.4, 7 Say oValor      VAR nValor Picture "@E 999,999,999,999.99"
      @1.9,.8 Say "Quantidade:"
      @1.9, 9 Say oQtda       VAR nQtdBem Picture "@E 99999" SIZE 50,10
      @1.4,15 Say Iif(MVParBox01==1,;
      "Somente grupos de ["+MVParBox02+"] ate ["+MVParBox03+"]","")
      If cTipoDet == "D"
            @12.4,01 Say "Qtde. a Detalhar"
            @158,60 MSGET nQtde Picture "@E 999" SIZE 036, 10 OF oDlg PIXEL;
            VALID nQtde > 0
      ElseIf cTipoDet == "P"
            @12.4,01 Say oLabel Var cLabel := "Tipo de Projeto"
      oRad  := TRadMenu():New(169,005,{"Industrial","Florestal"},;
               bSetGet(nOpcRad),oDlg,,{|| AF250RAD(nOpcRad)},,,,,,100,12,,,,.T.)
            @15.5,01 Say oLabel2 Var cLabel2 := "Detalhamento:"
      oRad2 := TRadMenu():New(210,005,{"Manter","Alterar"},;
               bSetGet(nOpcRad2),oDlg,,{|| AF250RAD2(nOpcRad2)},,,,,,;
               100,12,,,,.T.)
            @16.4,01 Say oLabel3 Var cLabel3 := "Percentual"
            oGet  := TGet():New(210,043,bSetGet(nPerc),oDlg,030,010,;
                                "@E 999.99",,,,,,,.T.)
            @18.2,01 Say oLabel4 Var cLabel4 := "Qtde. Det."
            oGet2 := TGet():New(235,040,bSetGet(nQtde),oDlg,030,010,;
                                "@E 999",,,,,,,.T.)
            oGet:Hide()
            oGet2:Hide()
            oLabel3:Hide()
            oLabel4:Hide()
      EndIf
      oMark := MsSelect():New("SN3","N3_OK","!N3_BAIXA",,@lInverte,;
                              @cMarca,{35,1,143,315})
      oMark:bMark := {| | a250Display(cMarca,lInverte,oValor,oQtda)}

      ACTIVATE MSDIALOG oDlg1
      ON INIT EnchoiceBar(oDlg1,;
      {|| nOpct:=1,iif(DeParaEnt(),oDlg1:End(),.f.)},{|| nOpct:=2,oDlg1:End()})
```

39. Introdução à relatórios gráficos

39.1. TReport()

39.1.1. Introdução

Finalidade

O Protheus oferece o recurso personalização para alguns relatórios de cadastros e movimentações do sistema. Ele tem como principais funcionalidades a definição de cores, estilos, tamanho, fontes, quebras, máscara das células para cada seção, criação de fórmulas e funções (Soma, Média, etc.), possibilidade de salvar as configurações por usuário e criação de gráficos.

Com a funcionalidade de Relatórios Personalizáveis, o usuário pode modificar os relatórios padrões, criando seu próprio layout.

Vale lembrar que nem todos os relatórios são personalizáveis. Por exemplo, relatórios que tenham layout pré-definidos por lei e formulários (boletos, notas-fiscais, etc) não poderão ser alterados.

Os relatórios personalizados são gravados com extensão .PRT, diferenciando-se dos relatórios padrões que recebem a extensão .###R.

Descrição

O TReport é uma classe de impressão que substitui as funções SetPrint, SetDefault, RptStatus e Cabec.

A classe TReport permite que o usuário personalize as informações que serão apresentadas no relatório, alterando fonte (tipo, tamanho, etc), cor, tipo de linhas, cabeçalho, rodapé, etc.

Estrutura do componente TReport:

- O relatório (TReport) contém 1 ou mais seções (TRSection);
- Uma seção (TRSection) pode conter 1 ou mais seções;
- A seção (TRSection) contém células pré-definidas e células selecionadas pelo usuário;
- A seção (TRSection) também contém as quebras (TRBreak) para impressão de totalizadores (TRFunction);
- Os totalizadores são incluídos pela seção que automaticamente inclui no relatório (TReport).

Pré-Requisitos

Para utilizar o TReport, verifique se o seu repositório está com o Release 4 do Protheus-8, ou versão superior.

A função TRepInUse() verifica se a lib do TReport está liberada no repositório em uso. O retorno é uma variável lógica.

```
#include "protheus.ch"

User Function MyReport()
Local oReport

If TRepInUse()    //verifica se a opção relatórios personalizáveis está
disponível
      Pergunte("MTR025",.F.)

      oReport := ReportDef()
      oReport:PrintDialog()
EndIf
Return
```

Verifique também o parâmetro MV_TReport. Para utilizar os relatórios personalizáveis, o parâmetro MV_TREPORT (tipo numérico) deve ser alterado no ambiente Configurador, conforme uma das opções que seguem:

1 = utiliza relatório no formato tradicional (antigo);

2 = utiliza relatório personalizável;

3 = pergunta qual relatório será utilizado: tradicional (antigo) ou personalizável.

39.1.2. Impressão do relatório personalizável

Cada componente da tela de impressão do TReport, deve ser configurado no programa, para que o usuário tenha acesso às personalizações:

39.1.2.1. Parâmetros de impressão

A caixa de listagem apresentada deve ser utilizada conforme o meio de saída do relatório. Veja a seguir.

Impressão

Arquivo

O relatório será gravado em disco com o nome apresentado. Caso seja escolhida a opção "Servidor" ele será gravado no diretório determinado na senha do usuário, através do configurador, sendo este sempre no servidor (padrão \SPOOL\). Na escolha da opção "Local" será aberta uma janela para que seja escolhido o local onde o relatório será gravado na máquina do usuário.

O relatório gerado a partir desta opção pode ser impresso ou enviado por e-mail após ser apresentado na tela.

Spool

Direciona o relatório para impressão via configuração do Windows® das impressoras instaladas.

E-mail

Envia o relatório por e-mail (Internet). Para isto, devem ser configurados os seguintes parâmetros no Ambiente Configurador:

- ☑ **MV_RELACNT**

  Define a conta de e-mail para identificar a proveniência dos relatórios. Exemplo: relprotheus@microsiga.com.br

- ☑ **MV_RELPSW**

  Define a senha da conta de e-mail para envio dos relatórios.

- ☑ **MV_RELSERV**

  Define o servidor da conta de e-mail para o envio do relatório. Exemplo: smtp.microsiga.com.br

Quando selecionada esta opção, deve-se informar, no campo em destaque na figura abaixo, o e-mail para o qual o relatório deve ser remetido.

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.

Papel

Tamanho do papel

Selecione o tamanho do papel em que o relatório será impresso.

As especificações de tamanho do papel são as do padrão do mercado, conforme o formato escolhido, o Protheus irá ajustar a impressão.

Formato da impressão

Selecione o formato de impressão, clicando nos botões de opção “Retrato” ou “Paisagem”, fazendo assim que o relatório seja impresso na orientação vertical ou horizontal, respectivamente.

Configurações

Título

Caso queira alterar a opção sugerida pelo sistema, digite o cabeçalho do relatório.

Ordem

Escolha a ordem em que as informações serão apresentadas no relatório, clicando em uma das chaves disponíveis.

Layout

Permite selecionar o modelo de relatório para impressão, à medida que novos leiautes forem gravados para um relatório, seus nomes serão listados nessa caixa.

Preview

Faz a exibição do relatório gerado na tela, possibilitando, na seqüência, o seu envio para impressora ou a cancelamento da impressão.

Executar em segundo plano

Essa opção permite que o relatório seja gerado e enviado para a fila de impressão, enquanto o usuário pode executar outras tarefas no sistema.

39.1.3. Personalização

É possível configurar-se as colunas do lay-out do relatório, bem como os acumuladores, cabeçalhos e linhas.

Estão disponíveis para personalização também a fonte, tamanho, cores, e etc.

39.1.3.1. Editando o layout do relatório

O primeiro passo é entender a nova estrutura dos relatórios desenvolvidos com a ferramenta TReport.

O Relatório possui Seções e Células. É chamada de Seção, cada um dos grupos de informações, e de Célula, cada um dos campos que serão impressos.

Nova estrutura do relatório TReport:

O relatório mais simples que se consegue emitir em TReport, é uma listagem.

39.1.4. Definindo a Função ReportDef()

A função ReportDef() é responsável pela construção do lay-out do relatório (oReport). É ela quem define as colunas, os campos e as informações que serão impressas.

Os comandos que fará essa construção são:

1. DEFINE REPORT
2. DEFINE SECTION
3. DEFINE CELL

DEFINE REPORT

A função DEFINE REPORT é responsável pela criação do objeto Report, ou seja, o relatório.

Internamente, o DEFINE REPORT irá executar o método TReport():New().

Estrutura do componente TReport:

- ⇒ O relatório (TReport) contém 1 ou mais seções (TRSection);
- ⇒ Uma seção (TRSection) pode conter 1 ou mais seções;
- ⇒ A seção (TRSection) contém células pré-definidas e células selecionadas pelo usuário;
- ⇒ A seção (TRSection) também contém as quebras (TRBreak) para impressão de totalizadores (TRFunction);
- ⇒ Os totalizadores são incluídos pela seção que automaticamente inclui no relatório (TReport).

DEFINE SECTION

Ainda no ReportDef(), são definidas as seções (oSection) do relatório.

As seções do relatório representam os diferentes grupos de informações exibidos.

Há a seção principal e as específicas.

Internamente, o DEFINE SECTION irá executar o método TRSection():New().

A classe TRSection pode ser entendida como um layout do relatório, por conter células, quebras e totalizadores que darão um formato para sua impressão.

Com a classe TRSection é possível definir uma query, filtro ou índice com filtro (IndRegua) que será utilizada por ela para processamento do relatório, através do método Print e utilizando as células de posicionamento (TRPosition).

DEFINE CELL

Para cada seção, devem ser definidas as células. Célula é cada informação que deverá ser impressa. Pode ser um campo do cadastro, ou um resultado de uma operação. É uma Célula de impressão de uma seção (TRSection) de um relatório que utiliza a classe TReport

Internamente, o DEFINE CELL irá executar o método TRCell():New ().

MÓDULO 08: Aplicações ADVPL para o ERP

40. Protheus e o TOPCONNECT / TOTVS DbAccess

O TOTVSDBAccess é uma ferramenta de conectividade a bases de dados, permitindo que aplicativos acessem uma ampla variedade de servidores de banco de dados sem a necessidade de geração de códigos específicos para cada uma delas.

Os bancos de dados hoje suportados pelo TOTVSDBAccess são:

- ☑ Microsoft SQL7® ou superior
- ☑ Oracle 8® ou superior
- ☑ IBM Universal Database® (DB2)
- ☑ Informix®
- ☑ Sybase Adaptive Server®
- ☑ Postgres 7.1.1® ou superior
- ☑ MySQL®

O TOTVSDBAccess permite gerenciar as informações pertinentes à utilização do banco de dados, registrando e apresentado dados, como:

- ☑ Dados estatísticos de forma gráfica com relação à:
- ☑ Quantidade de usuários conectados
- ☑ Quantidade de tabelas em utilização
- ☑ IOs por segundo
- ☑ Máximo de IOs por segundo
- ☑ Relação dos usuários conectados
- ☑ Checagem das querys geradas pelos usuários conectados
- ☑ Quebra de conexão do usuário
- ☑ Mensagens de utilização do TOTVSDBAccess
- ☑ Configurações Gerais quanto a:
  - ❒ Banco de dados em utilização
  - ❒ Relação de usuários e senhas por banco de dados
  - ❒ Logins de usuário
  - ❒ Configuração de "Table Spaces" para determinados bancos de dados
  - ❒ Definição dos parâmetros de criação de tabelas para determinados bancos de dados
  - ❒ Validação de conexão, checando os parâmetros de conexão com o banco de dados
  - ❒ Monitoramento de índices gerados

40.1. Características técnicas com o uso do TOTVS DbAccess

Comportamento de Queries – Colunas Calculadas

Após enviar ao Banco de Dados uma solicitação de abertura de Query, em caso de sucesso na operação, o TOPConnect obtém do Banco de Dados a quantidade de colunas que serão retornadas, e qual é o tamanho e tipo e especificações de cada uma das colunas, para retornar ao Protheus uma estrutura definida de informações, e o cursor aberto é lido conforme são solicitados os próximos registros (DBSkip() no Advpl), e a cada requisição, uma linha do cursor é retornada, até não haver mais linhas no cursor, atingindo o final do arquivo no Advpl (EOF()).

Quando especificamos uma coluna de retorno na Query, que corresponde a informações lidas diretamente de uma ou mais coluna(s) da tabela, cada coluna retornada pela Query contém uma definição de tipo e tamanho correspondente ao campo de origem do dado de retorno. Deste modo, se considerarmos, por exemplo, a Query : SELECT A1_FILIAL FROM SA1990 , onde o campo A1_FILIAL na tabela SA1990 é um campo do tipo varchar, com 2 bytes de tamanho, quando o cursor for aberto no Banco de Dados, o TOPConnect irá obter do banco que o result-set gerado possui apenas uma coluna, e esta coluna é do tipo varchar, com no máximo 2 bytes de tamanho.

Os Bancos de Dados não apenas permitem recuperar retornos diretamente lidos de uma ou mais tabelas, mas também é possível abrir um cursor onde uma ou mais colunas contenham um resultado calculado, utilizando-se de expressões e/ou funções nativas do Banco de Dados. Para estes tipos de retorno, cada banco de dados pode apresentar comportamentos diferenciados ao descrever o tamanho de retorno da(s) coluna(s).

Este comportamento pode variar de acordo com o Banco de Dados, a expressão e/ou funções utilizadas para gerar a coluna de retorno, e inclusive podem ter diferenças entre versões e Build's do mesmo Banco de Dados e/ou parametrizações específicas do Banco de Dados e/ou da aplicação Client / ODBC / API utilizada para acesso ao Banco.

Para ilustrar melhor estas diferenças, vejamos abaixo os resultados obtidos com queries, em bancos distintos, onde foi criada uma tabela com a mesma estrutura, e 10 linhas de dados, e nela foram executadas algumas querys, onde temos como retorno uma coluna calculada. A tabela possui os campos CPOC, tipo Caractere, tamanho 10 bytes, numerados de 0000000001 a 0000000010 e um campo numérico CPON, alimentado com valores positivos. Para cada Query, mostramos o valor retornado ao Advpl, contendo o tipo de dado retornado e o tamanho (len) da coluna, no formato . (Tipo (tamanho)).

Query: *SELECT MIN(CPOC) CPOTESTE FROM TESTRET*

- ☑ **Retorno:** Todos os bancos retornaram uma coluna com tamanho de 10 bytes.
- ☑ **Bancos:**

Database	Tipo	Tamanho	Conteúdo
MSSQL 2000	Caracter	10	0000000001
INFORMIX	Caracter	10	0000000001
ORACLE	Caracter	10	0000000001
MYSQL	Caracter	10	0000000001
DB2	Caracter	10	0000000001

Query: *SELECT MIN(CPOC) CPOTESTE FROM TESTRET WHERE CPON < 0*

☑ **Retorno:** Todos os bancos retornaram uma coluna vazia.

☑ **Bancos:**

Database	Tipo	Tamanho	Conteúdo
MSSQL 2000	Caracter	10	Vazio
POSTGRES	**Caracter**	**0**	**Vazio**
INFORMIX	Caracter	10	Vazio
ORACLE	Caracter	10	Vazio
MYSQL	Caracter	10	Vazio
DB2	Caracter	10	Vazio

Query: *SELECT LEFT(CPOC,5) CPOTESTE FROM TESTRET*

☑ **Retorno:** Todos os bancos que suportam a função LEFT() retornaram uma coluna de informações, onde os 5 primeiros bytes eram '00000'.

☑ **Observações:** Os bancos retornaram que a coluna tem 5 bytes de tamanho. Porém, o BAnco DB2 informa que a coluna têm tamanho de 4000 Bytes. Este mesmo retorno foi obtido no DB2 quando utilizadas funções RIGHT(), REPLACE() e REPEAT(), no banco DB2, e para o BAnco Postgres, foi retornado o tamanho de 2000 bytes para estas funções.

☑ **Bancos:**

Database	Tipo	Tamanho	Conteúdo
MSSQL 2000	Caracter	5	00000
MYSQL	Caracter	5	00000
DB2	**Caracter**	**4000**	**00000 ...**

Query: *SELECT SUBSTRING(CPOC,2,4) CPOTESTE FROM TESTRET*

☑ **Retorno:** Todos os bancos retornaram uma coluna de tamanho 4 bytes, exceto o POSTGRES, que retornou tamanho de 2000 bytes.

☑ **Bancos:**

Database	Tipo	Tamanho	Conteúdo
MSSQL 2000	Caracter	4	0000
POSTGRES	Caracter	2000	0000 ...
INFORMIX	Caracter	4	0000
ORACLE	Caracter	4	0000
MYSQL	Caracter	4	0000
DB2	Caracter	4	0000

Devido a estas diferenças de comportamento, devemos tomar alguns cuidados ao desenvolver aplicações que utilizam colunas calculadas em queries. Nestes casos, deve ser utilizada uma função do banco de dados, para assegurar que o banco de dados retorne o tamanho de coluna adequado ao TOPConnect / TotvsDBAccess.

Por exemplo, vejamos a query abaixo :

```
SELECT REPLACE(CPOC,'0','_') AS CPOTESTE FROM TESTRET
```

Se já é de conhecimento do programador que o campo da base de dados CPOC desta tabela tem 10 bytes de tamanho, devemos fazer um cast deste retorno, para char ou varchar, com tamanho de 10 bytes. Dessa forma, o banco retorna o tamanho esperado da coluna para o TOPConnect quando da abertura da query. Isto evita que espaços desnecessários trafeguem pela rede para atender à requisição, e que este valor excedente seja usado para alimentar uma variável do Advpl. Vejamos como usar este recurso na query acima descrita, utilizando uma sintaxe aceita pela maioria dos Bancos de Dados homologados :

```
SELECT CAST( REPLACE(CPOC,'0','_') AS VARCHAR(10)) CPOTESTE FROM TESTRET
```

Vale a pena salientar que, cada banco de dados possui, com pequenas variações, uma sintaxe para permitir o CAST de um retorno. Para maiores detalhes, a documentação do banco deve ser consultada.

Comportamento diferenciado com Bandos de Dados PostGres

O Banco de Dados Postgres possuem um comportamento diferenciado entre os Builds do Server do Banco, e possui uma parametrização na configuração do ODBC que pode interferir na maneira como os atributos das colunas de retorno de uma query são retornados.

Em versões inferiores ào Postgres 8.x, as diferenças de comportamento em relação àos demais Bancos de Dados possuem um diferencial muito significativo, em praticamente todas as funções de retorno calculado. Para estes casos, em se tratando de Postgres versões inferiores a 8.x, a precisão de retorno da coluna corresponde ao tamanho máximo de retorno do tipo VarChar. (Este tamanho máximo pode ser configurado na conexão ODBC do PostgreSql, e o valor default é 65536 bytes.)

Na função Substring(), mesmo que esteja especificado no 3. parâmetro da função, a quantidade final limitada de Bytes, o Banco de Dados descreve esta coluna de retorno como sendo um VarChar() com a precisão do maior valor VARCHAR() a ser retornado pelo client. Na função Max(), quando especificado um campo da base do tipo Char ou Varchar, temos um comportamento semelhante.

Quando foram homologados os Bancos Postgres para uso com o ERP Microsiga e TOPConnect 4, praticamente não eram utilizadas queries com colunas calculadas nas aplicações Advpl, e este comportamento diferenciado não foi percebido. Conforme as rotinas foram evoluindo, buscando mais performance através do uso de Queries, tornou-se perceptível estas diferenças operacionais.

Algumas destas diferenças operacionais foram assimiladas pelas funções do ERP, por exemplo a ChangeQuery, que em versão atualizada (** Protheus 8 e P10, fonte aplib070.prw com data igual ou superior a 08/03/2008) , aplicam o CAST() automaticamente para SELECT MAX(CAMPO), quando o banco é Postgres, e o campo selecionado for um campo da base de dados, presente no Dicionário de Campos (SX3).

Para todos os efeitos, para utilização com as versões homologadas de Postgres inferiores a 8.x, é necessário alterar a configuração de ODBC do Postgres, para limitar o retorno do tipo VarChar para 2000 Bytes, pois o retorno do valor default (65536 bytes) não é suportado pelo TOPConnect / TOTVSDBAccess. Alteramos esta configuração no arquivo .odbc.ini no linux, e/ou no arquivo de configuração do ODBC do Postgres utilizado, inserindo na seção da conexão / identificação do banco a chave abaixo :

```
MaxLongVarcharSize=2000
```

Atualmente está em processo de homologação as versões mais recentes (8.x) do banco Posgres. Quando homologado, a informação estará disponivel na DEM, em Principal -> Guia de Referência -> Guia de Plataformas Homologadas -> Bancos de Dados Relacional X S.O.

Conceito de Índices Permanentes e Diferenças das RDDs

Na RDD TOPCONN, não temos o conceito de criação de índice temporário, apenas índice permanente. Um índice permanente é criado fisicamente no Banco de Dados relacional utilizado, através do TOPConnect / DbAccess, onde devemos especificar um ou mais campos para compor a chave / expressão de índice.

Quando utilizamos CodeBase/DBF, podemos especificar como campos de chave de índice o retorno de funções, como por exemplo STR(), DTOS(), SUBSTR(), entre outras suportadas de modo nativo pela aplicação provedora de acesso, no caso ADS Local e/ou ADS Server.

Quando usamos um Banco Relacional SGDB através do TOPConnect / DbAccess, devemos especificar uma expressão de chave que sempre retorne um valor Caractere, e as únicas funções permitidas para adequações de tipo de dado no Advpl para estas expressões de índice são : DTOS() e STR().

Funcionamento Interno

A expressão utilizada para a criação de um índice permanente em Advpl, por exemplo : CPOC + DTOS(CPOD) + STR(CPON,10), quando utilizamos o RDD TOPCONN, será ajustada para o Banco de Dados relacional utilizado para uma expressão contendo apenas a lista de campos desejada, na sequência em que foi especificada, e as funções DTOS() e STR() nao são passadas ao banco de dados, pois não existe a necessidade de conversão de dados para o Banco.

Um campo do tipo 'D' Data é tratado pelo TOPConnect e gravado na tabela em questão como um campo 'C' Caractere, de 10 bytes, no formato AAAAMMDD, e os números são gravados em um campo DOUBLE*.

Quebra de Compatibilidade com CodeBase / DBF

Os Bancos relacionais, em sua grande maioria, senão todos, suportam apenas a criação de índices onde sejam especificados campos físicos da base de dados. Não são suportadas funções de conversão ou transformação de dados na criação de índices, como por exemplo substring(), left(), entre outras. Embora alguns bancos permitam a criação de colunas calculadas, as mesmas são 'virtuais', isto é, não existem fisicamente na tabela, e também não é permitido o uso de colunas virtuais para a criação de índices.

Entendemos que o RDD CodeBase / DBF, onde originalmente foi desenvolvido o ERP, nos dava certa flexibilidade ao permitir operações como estas, porém a um custo de processamento mais alto, pois para cada inserção ou alteração, o RDD tem que executar funções de conversao e concatenação de strings para atualização dos índices que usam este recurso.

Manter o suporte a todos os indices permanentes, criados a partir de resultados de expressões, nas versões anteriores do ERP, teríamos um custo muito alto de complexidade, performance e duplicidade de informações nas tabelas. Seria necessário criar colunas físicas nas tabelas, transparentes ao usuario, para manter copias em duplicidade de partes de colunas agrupadas, com gatilhos do banco de dados disparados internamente em operações de insert e update, um mecanismo relativamente complexo de se manter, instável para dar manutenção, e custoso em performance para o Banco de Dados.

Partindo da premissa que, se uma determinada informação deve ser indexada para busca, ela deve ser uma informação que ocupa exclusivamente um campo físico da base de dados, são evitadas as operações constantes de concatenação e desmembramento de uma informação agrupada, colocando cada parte da informação em seu devido espaço (campo) definido, mesmo com o impacto gerado para alterar as aplicações que usavam estes tipos de índices, os ganhos obtidos em organização e performance foram muito significativos.

Lista dos códigos de erro do TOPConnect / DbAccess

Quanto utilizado o TOPConnect / DbAccess para conexão e operações com Bancos de Dados, as ocorrências de erro são reportadas ao Protheus informando em conjunto com o erro um número menor que zero, correspondendo a um tipo de erro.

Para a maioria das ocorrências, relacionadas ao Banco de Dados, deve-se olhar o log de erro gravado pelo TOPConnect / DbAccess (arquivo topconn.log) para obter maiores detalhes sobre a ocorrência.

TC_NO_ERROR	0	**COMM_INITPGM_ERROR**	-81
NO_ROUTER_INSTALLED	-1	**COMM_PARAM_ERROR**	-86
NO_CONNECTION	-2	**COMM_PROGRAM_ERROR**	-88
NO_USER_SECURITY	-4	**COMM_INSMEM_ERROR**	-90
PASSTHRU_FAILED	-5	**INVALID_BUILD**	-99
NO_MORE_CONNECTIONS	-6	**INVALID_TOPAPI**	-100
INVALID_TOP_KEY	-8		
INVALID_ENVIRONMENT	-9		
INVALID_FILE	-10		
UNKNOWN_FILE	-11		
EXCLUSIVE_REQUIRED	-11		
INVALID_OPERATION	-13		
INVALID_KEY_NUM	-14		
FILE_IN_USE	-15		
TOO_MANY_FILES	-16		
INVALID_NUMRECS	-17		
CALL_FAILED	-18		
COMMAND_FAILED	-19		
OVERRIDE_FAILED	-20		
QUERY_FAILED	-21		
CREATION_FAILED	-22		
OPEN_FAILED	-23		
NOT_OPENED	-24		
NO_RECORD_FOUND	-25		
END_OF_RECORDS	-26		
NO_WRITE_POSIBLE	-27		
NO_RECORD_EQUAL	-28		
UPDATE_FAILED	-29		
DELETE_FAILED	-30		
RECORD_LOCKED	-31		
FILE_LOCKED	-32		
NO_AUTORIZATION	-33		
TOO_MANY_USERS	-34		
NO_DB_CONNECTION	-35		
NO_CONN_ALLOWED	-36		
INTEGRITY_FAILURE	-37		
BUFFER_OVERFLOW	-40		
INVALID_PARAMETERS	-41		
NO_AUDIT_CONNECTION	-50		
COMM_DOSMEM_ERROR	-58		
COMM_PARTNER_ERROR	-67		
COMM_SNDSTAT_ERROR	-76		
COMM_RCVSTAT_ERROR	-76		

40.2. Funções ADVPL para TOPCONNECT / TOTVS DbAccess

Neste tópico serão descritas as funções da linguagem ADVPL que permitem a interação com a aplicação TopConnect / DbAccess.

Lista das funções de interação com a aplicação TopConnect / DbAccess:

- **TCCANOPEN**
- **TCCONTYPE**
- **TCDELFILE**
- **TCGENQRY**
- **TCGETDB**
- **TCLINK**
- **TCQUERY**
- **TCQUIT**
- **TCSETCONN**
- **TCSETFIELD**
- **TCSPEXEC**
- **TCSPEXIST**
- **TCSQLERROR**
- **TCSQLEXEC**
- **TCSRVTYPE**
- **TCUNLINK**
- **TCCHKOBJ**
- **TCEXEERROR**
- **TCPGMEXE**
- **TCSYSEXE**

Lista das funções acessórias utilizadas nos fontes como facilitadoras:

- **CHANGEQUERY**
- **RETFULLNAME**
- **RETSQLCOND**
- **RETSQLNAME**
- **RETSQLTABLE**
- **SQLCOPY**
- **SQLORDER**
- **SQLTOTRB**

Funções de interação com o TopConnect / DbAccess

TCCANOPEN ()

Verifica a existência de tabelas e índices no servidor.

☑ **Sintaxe: TCCanOpen(cTable , cIndice)**

☑ **Parâmetros:**

cTable	Nome completo da tabela no banco de dados.
cIndice	Opcional, indica o nome completo do índice referente a tabela especificada.

☑ **Retorno:**

Lógico	Indica a existência da tabela e/ou do índice especificado.

☑ **Exemplo:**

```
//Verifica a existência da tabela customer
IF !TCCanOpen("CUSTOMER")
   dbCreate("CUSTOMER", aStru, "TOPCONN")
ENDIF
USE CUSTOMER SHARED NEW VIA "TOPCONN"

//Verifica a existência do índice
IF !TCCanOpen("CUSTOMER","CUSTCOD")
    INDEX ON CODIGO TO CUSTCOD
ELSE
   SET INDEX TO CUSTCOD
ENDIF
```

TCCONTYPE()

Especifica o protocolo de comunicação a ser utilizado pelo TOPConnect.

☑ **Sintaxe: TCConType(cType)**

☑ **Parâmetros:**

cType	Tipo de protocolo de comunicação a ser utilizado pelo TopConnect, aonde: TCPIP: Utiliza o protocolo TCP/IP. NPIPE: Utiliza o protocolo Named Pipes. APPC: Utiliza o protocolo APPC. BRIDGE: Utiliza o utilitário TOPBrigde para comunicação de estações DOS NPIPE com servidor AS/400 utilizando TCP/IP.

☑ **Retorno:**

Nenhum	.

☑ **Exemplo:**

```
#INCLUDE "TOPCONN.CH"
//Especifica conexão TCP/IP
TCConType("TCPIP")

//Conecta-se ao ambiente SIGAADV no Microsoft SQL-Server
TCLink("MSSQL/SIGAADV")
```

TCDELFILE()

Exclui uma tabela no servidor.

☑ **Sintaxe: TCDelFile(cTable)**

☑ **Parâmetros:**

cTable	Nome completo da tabela a ser excluída.

☑ **Retorno:**

Nenhum	.

☑ **Exemplo:**

```
TCDelFile("CUSTOMER")
```

Anotações

TCGENQRY()

Permite a execução de uma query de seleção no banco de dados e retorna um recordset com o resultado.

☑ **Sintaxe: TCGENQRY(xParam1,xParam2,cQuery)**

☑ **Parâmetros:**

xParam1	Parâmetro reservado.
xParam2	Parâmetro reservado.
cQuery	Query de seleção com sintaxe no padrão SQL ANSI a ser executada no banco de dados.

☑ **Retorno:**

RecordSet	RecordSet contendo o resultado da execução da query de seleção.

Importante

Para que os dados contidos no RecordSet retornado pela função TCGenQry() sejam utilizados pela aplicação ADVPL é necessário disponibilizá-los em uma WorkArea.

Desta forma a função TCGenQry() deverá ser utilizada em conjunto com a função DbUseArea(), para que a mesma disponibilize o RecordSet como uma área de trabalho do ADVPL.

☑ **Exemplo:**

```
cQuery := "SELECT DISTINCT CV8_PROC FROM "+RETSQLNAME("CV8")
cQuery += " WHERE "
cQuery += "CV8_FILIAL = '"+MV_PAR01+"' AND "
cQuery += "D_E_L_E_T_ = '' "
cQuery += "ORDER BY CV8_PROC"
cQuery := ChangeQuery(cQuery)
dbUseArea(.T., "TOPCONN", TCGenQry(,,cQuery), "CV8QRY", .F., .T.)
```

TCGETDB()

Retorna o tipo de Banco de Dados corrente.

☑ **Sintaxe: TCGETDB()**

☑ **Parâmetros:**

Nenhum	.

☑ **Retorno:**

Caracter	Descrição referente ao tipo do banco de dados corrente, aonde: ANYWHERE - Sybase SQL Anywhere ANYSYBASE - Sybase SQL Anywhere (Emulando Sybase SQL-Server) DB2 - IBM Universal Database INFORMIX – Informix MSSQL - Microsoft SQL-Server 6.5 MSSQL7 - Microsoft SQL-Server 7.0 ou superior ORACLE - Oracle POSTGRES - PostGres SYBASE - Sybase SQL-Server Nota: O retorno da função pode variar de acordo com a homologação de novos databases para utilização com a ferrramenta TopConnect / DbAccess.

☑ **Exemplo:**

```
cDataBase := TCGETDB() // "MSSQL7"
```

TCLINK()

Abre uma conexão com o Servidor TOPConnect.

☑ **Sintaxe: TCLink(cConectString,cServer)**

☑ **Parâmetros:**

cConectString	String composta conforme a regra: Para TOPConnect NT : Nome do Banco de Dados / Nome do Ambiente Para TOPConnect/400: Nome do Ambiente
cServer	Nome do servidor conforme a regra: Para conexões NamedPipe, APPC e Brigde : O nome do servidor TOPConnect. Para conexões TCP/IP : O endereço IP do servidor TOPConnect

☑ **Observações:**

- ❒ Para o TOPConnect NT acessando Oracle, um ambiente é o nome do Alias criado com o utilitário SQL-Net.
- ❒ Para o TOPConnect NT acessando os demais SGDB's um ambiente é um System DSN criado através do ODBC Data Source Administrator.
- ❒ Para o TOPConnect AS/400 um ambiente é uma Library criada através do comando CRTTOPENV do TOPConnect.

- Para informações sobre a criação de ambientes consulte o manual de instalação do TOPConnect.

- A lista dos tipos de bancos de dados suportados pela aplicação TopConnect / DbAccess é a mesma das possibilidades de retorno da função TCGETDB conforme abaixo:

ANYWHERE	- Sybase SQL Anywhere
ANYSYBASE	- Sybase SQL Anywhere (Emulando Sybase SQL-Server)
DB2	- IBM Universal Database
INFORMIX	– Informix
MSSQL	- Microsoft SQL-Server 6.5
MSSQL7	- Microsoft SQL-Server 7.0 ou superior
ORACLE	- Oracle
POSTGRES	- PostGres
SYBASE	- Sybase SQL-Server

☑ **Retorno:**

Numérico	A função TCLink retorna ou o número da conexão ou um valor negativo contendo o código do erro. A relações dos códigos de erros ocorridos durante uma tentativa de conexão está disponível no tópico 8.1. deste material.

☑ **Exemplo:**

```
//Conecta-se ao Microsoft SQL-Server no ambiente SIGAADV
//Protocolo Named Pipes
TCConType("NPIPE")

nCon := TCLink("MSSQL/SIGAADV","TOPSRV")
if nCon < 0 //Conexões com retorno < 0 significam erro
   Alert("Falha de conexão com o TOPConnect")
endif
//Protocolo TCP/IP
TCConType("TCPIP")
//Conecta-se ao Oracle – no ambiente TESTES
nCon := TCLink("ORACLE/TESTES",172.16.1.2)

//Protocolo APPC
//Conecta-se ao AS/400 no ambiente PRODUCAO
nCon := TCLink("PRODUCAO","TOP400")
```

Anotações

TCQUERY()

Executa uma Query no servidor e coloca seu retorno em uma WorkArea.

Durante o processo de compilação, a sintaxe TCQUERY() é substituída pelas expressão:

dbUseArea(.T.,"TOPCONN",TcGenQry(,,cQuery), “ALIAS” ,.T.,.F.)

Esta substituição é realizada conforme as definições do include TOPCONN.CH.

Desta forma é recomendável a utilização direta da sintaxe DbUseArea() + TcGeQry().

☑ **Sintaxe: TCQUERY cSQLExpr ALIAS cAlias NEW VIA "TOPCONN"**

☑ **Parâmetros:**

cSQLExpr	Expressão SQL a ser enviada ao servidor.
ALIAS cAlias	Especifica um nome para a Work Area a ser aberta.
NEW	Abre a tabela <cTable> na próxima Work Area disponível. Se esta clausula não for especificada, <cTable> será na Work Area corrente.
VIA "TOPCONN"	Este parâmetro indica ao ADVPL que esta Work Area será gerenciada pelo TOPconnect.

☑ **Observações:**

- A Work Area criada com o comando TCQUERY é READ-ONLY, portanto não é permitido o uso de APPEND's ou REPLACE's.
- O TOPConnect tem apenas um cursor para a Query executada portando apenas os seguintes comandos são suportados em uma Work Area criada com TCQUERY:
  - GO TOP - DbGoTop()
  - GO BOTTOM - DbGoBottom()
  - SKIP - DbSkip()
  - Eof()
  - Bof()
- Ao executar uma Query que retorne um ou mais valores calculados, o nome dos campos da WorkArea serão COLUMN1, COLUMN2... COLUMNn.

☑ **Retorno:**

Nenhum	.

☑ **Exemplo:**

```
//Abre a tabela de Clientes em uma nova WorkArea

cQuery := "SELECT a.codigo, b.nome FROM CLIENTES a, CLIDATA b "
cQuery += "WHERE a.CODIGO = b.CODIGO ORDER BY CODIGO,NOME "

TCQUERY cQuery ALIAS CLIENTES NEW VIA "TOPCONN"
dbGoTop()

While !Eof()
   ALERT(CODIGO+"|"+NOME)
   dbSkip()
end

cQuery := 'SELECT SUM(VALOR),SUM(SALDO) FROM CUSTOMER'
TCQUERY cQuery NEW VIA "TOPCONN"
ALERT(CValToChar(COLUMN1)+"|"+CValToChar(COLUMN2))
// COLUMN1 - Somatoria dos valores
// COLUMN2 - Somatoria dos saldos
```

TCQUIT()

Encerra todas as conexões com o TOPConnect.

☑ **Sintaxe: TCQuit()**

☑ **Parâmetros:**

Nenhum	.

☑ **Retorno:**

Nenhum	.

TCSETCONN()

Seleciona a conexão ativa.

☑ **Sintaxe: TCSETCONN(nConn)**

☑ **Parâmetros:**

nConn	Número da conexão

☑ **Retorno:**

Nenhum	.

☑ **Exemplo:**

```
// Abre conexão com o ambiente de Produção
nCon1 := TCLink("MSSQL/PRODUCAO")
if nCon1 < 0
   Alert("Falha conectando ambiente de Produção")
   QUIT
endif

// Abre conexão com o ambiente de Testes
nCon2 := TCLink("MSSQL/TESTES")

if nCon2 < 0 then
   Alert("falha conectando ambiente de Testes")
   QUIT
endif

TCSetConn(nCon1)

//Abre tabela de Clientes no ambiente de produção
USE CUSTOMER ALIAS PROD SHARED NEW VIA "TOPCONN"

TCSetConn(nCon2)

//Abre tabela de Clientes no ambiente de testes
USE CUSTOMER ALIAS TEST SHARED NEW VIA "TOPCONN"
...
```

TCSETFIELD()

Esta função serve como apoio ao comando TCQUERY, na recuperação de campos tipo NUMERIC, DATE e LOGICAL, pois os mesmos são gravados fisicamente no Banco de Dados como caracteres, e no caso dos numéricos como float.

☑ **Sintaxe: TCSetField(cAlias, cField ,cType, nTam, nDec)**

☑ **Parâmetros:**

cAlias	Alias da WorkArea gerada pela execução da query.
cField	Nome do campo a ser tratado
cType	Tipo desejado para o campo
nTam	Tamanho total desejado para o campo
nDec	Número de decimais desejado para o campo

☑ **Retorno:**

Nenhum	.

☑ **Exemplo 01:**

```
TCQUERY "SELECT NOME, DATA, MARRIED, VALOR FROM CUSTOMER" ALIAS QUERY VIA
"TOPCONN"

TCSetField("QUERY","DATA","D")
TCSetField("QUERY","VALOR","N",12,2)
```

Dentro de um programa ADVPL já podemos considerar como um campo data não mais um campo caracter como é o seu armazenamento.

O campo Data só é retornado como um campo caracter por que estamos utilizando a função TCQUERY, se não o tratamento é automático.

A estrutura dos parâmetros da função TCSETFIELD() é a mesma retornada pela função DbStruct() e que é utilizada em diversas funções que manipulam arquivos de dados.

Desta forma é muito comum a montagem de queries em tabelas do sistema partindo da estrutura de campos retornada pela função DbStruct() sendo que após a execução dos campos Tcquery() ou DbUseArea() normalmente é utilizado o campo TcSetField com base no conteúdo deste mesmo array de estrutura.

☑ **Exemplo 02:**

```
Local cQuery      := ""
Local cCampos     := ""
Local aStruSA1    := SA1->(DbStruct())
Local nX          := 0
Local nMax        := Len(aStruSA1)

// Monta a String cCampos de acordo com o conteúdo de aStruSA1

aEval(aStruSA1,{|aCampo| nX++, cCampos += aCampo[1] +;
      IIF(nX == nMax,' ',', ')})

// Monta a query utilizando a String cCampos

cQuery := "SELECT "+cCampos+" FROM "+RetSqlName("SA1")

If Select("SA1SQL") <> 0
      DbSelectArea("SA1SQL")
      DbCloseArea()
Endif

dbUseArea(.T.,"TOPCONN",TcGenQry(,,cQuery),"SA1SQL",.T.,.F.)
```

Continuação:

```
For nX := 1 to Len(aStruSA1)
      IF !( aStrutSA1[nX][2] $ "C/M")
            TCSetField( cAlias,aStruSA1[nX][1],aStruSA1[nX][2],;
                        aStruSA1[nX][3],aStruSA1[nX][4])
      ENDIF
Next nX
```

TCSPEXEC()

Executa uma Stored Procedure no Banco de Dados.

Devido a uma limitação em alguns dos Bancos de Dados suportados na obtenção dos tipos de parâmetros (se são de INPUT e/ou OUTPUT) todos as Stored Procedures a serem executadas através do TOPConnect deverão obedecer o seguinte padrão de nomenclatura de seus parâmetros :

Parâmetros de INPUT devem começar com IN_... Ex. IN_VALOR.
Parâmetros de OUTPUT devem começar com OUT_... Ex. OUT_CODIGO

Após a execução de uma Stored Procedure o TOPConnect retornará ao ADVPL um array com 'n' elementos, onde n é o número de parâmetros de OUTPUT da Stored Procedure.

☑ **Sintaxe: TCSPExec(cSPName,xParam1,xParam2...xParamN)**

☑ **Parâmetros:**

cSPName	Nome completo da Stored Procedure no banco de dados.
xParamN	Parâmetros de input da Stored Procedure.

As procedur es padrões da aplicação ERP Protheus possuem concatenadas em seus nomes o código da empresa para a qual foram aplicadas.

Desta forma ao executar uma procedure padrão do ERP através do comando TCSPEXEC é necessário utilizar a função xProcedures(), pois a mesma irá concatenar ao nome da procedure que será executada o código da empresa.

Por exemplo: Procedure MAT001

- ❒ Se for executada com a sintaxe: TCSPExec("MAT001",...) ocorrerá um erro na aplicação pois esta procedure não existe.
- ❒ Utilizando a função xProcedures() temos:

TCSPExec(xProcedures("MAT001"), ...), onde caso a empresa em uso seja a 01, será executada a MAT001_01, a qual é válida.

☑ **Retorno:**

Array	Array com os parâmetros de retorno da Stored Procedure e <cSPName> é o nome da Stored Procedure a ser executada e os demais parâmetros variam conforme a definição da Stored Procedure.

☑ **Exemplo:**

```
//Verifica se a Stored Procedure Teste existe no Servidor
If TCSPExist("TESTE")

   //Executa a Stored Procedure Teste
   aRet := TCSPExec("TESTE","JOSE",1000)
   if aRet <> nil

      cRetorno := ""
      For i:= 1 to Len(aRet)
            cRetorno += "|"+ IIF(ValType(aRet[i]) == "C", aRet[i],;
                        IIF(ValType(aRet[i]) == "N", CValToChar(aRet[i]), "U"))
      Next

      //Mostra os valores de retorno
      ALERT(cRetorno)

   Else
      ALERT("Erro executando Stored Procedure"+CRLF+"Mensagem: "+TCSQLError())
   Endif
EndIf
```

Importante

A função TCSPExist() deve ser utilizada apenas para verificação da existência de Stored Procedures padrões da aplicação ERP, ou que possuam concate nadas no nome a informação da empresa para qual estão habilitadas.

Para verificação de procedures sem esta característica deve ser utilizada a função ExistProc().

Anotações

TCSPEXIST()

Verifica a existência de uma determinada Stored Procedure no servidor.

☑ **Sintaxe: TCSPExist(cSPName)**

☑ **Parâmetros:**

CSPName	Nome da procedure a ser verificada no banco de dados.

☑ **Retorno:**

Lógico	Indica se a Stored Procedure existe ou não e <cSPName> é o nome da Stored Procedure procurada.

☑ **Exemplo:**

```
If TCSPExist("MAT001_01")
   TCSPExec("MAT001_01",...)
Endif
```

A função TCSPExist() deve ser utilizada apenas para verificação da existência de Stored Procedures padrões da aplicação ERP, ou que possuam concate nadas no nome a informação da empresa para qual estão habilitadas.

Para verificação de procedures sem esta característica deve ser utilizada a função ExistProc().

TCSQLERROR()

Retorna o último erro registrado pelo TOPConnect durante a execução de uma Query.

☑ **Sintaxe: TCSQLError()**

☑ **Parâmetros:**

Nenhum	.

☑ **Retorno:**

String	Mensagem de erro registrada no TopConnect para a ocorrência.

☑ **Exemplo:**

```
If  TCSQLExec("UPDATE CUSTOMER SET VALUE=0") < 0 then
   ALERT(TCSQLError())
Endif
```

Importante

Esta é a mesma mensagem que esta registrada no log de eventos do TopConnect / TOTVS DbAccess Manager.

TCSQLEXEC()

Permite a execução de comandos de atualização no servidor de banco de dados.

☑ **Sintaxe: TCSQLExec(cCommand)**

☑ **Parâmetros:**

cCommand	Comando SQL a ser executado.

☑ **Retorno:**

Numérico	Retorna um valor negativo em caso de erros.

☑ **Exemplo:**

```
nRet := TCSQLExec("UPDATE CUSTOMER SET VALUE=0)
```

TCSRVTYPE()

Retorna o tipo do servidor no qual TOPConnect / TOTVS DbAccess está em execução.

☑ **Sintaxe: TCSrvType()**

☑ **Parâmetros:**

Nenhum	.

☑ **Retorno:**

String	Tipo do servidor, como por exemplo: "WinNT" ou "AS/400".

☑ **Exemplo:**

```
TCLink("MSSQL/TESTE","TOPSRV")
ALERT(TCSrvtype())
```

Anotações

Importante

Devido aos tratamentos específicos necessários nas queries que serão executadas em bancos de dados DB2 com sistemas operacionais AS/400 é normal a seguinte verificação:

```
#IFDEF TOP
        If TcSrvType() != "AS/400"

                <Tratamento SQL convencional>

        Else

#ENDIF

                <Tratamento padrão CODBASE>

#IFDEF TOP
        Endif
#ENDIF
```

TCUNLINK()

Encerra uma conexão com o TOPConnect.

☑ **Sintaxe: TCUnlink(nConn)**

☑ **Parâmetros:**

nConn	Número da conexão previamente retornado pela função TCLink()

☑ **Retorno:**

Nenhum	.

☑ **Exemplo:**

```
TCConType("NPIPE")
nConn := TCLink("MSSQL/TOPCONN","TOPSRV")
TCUnLink(nConn)
```

Anotações

TCCHKOBJ()

Verifica a existência de um objeto no servidor AS/400

☑ **Sintaxe: TCChkObj(cObj,cLibrary,cType)**

☑ **Parâmetros:**

cObj	Nome do objeto a ser verificado.
cLibrary	Nome da biblioteca que deve conter o objeto
cType	Tipo do de objeto AS/400, como por exemplo: *FILE ou *PGM.

☑ **Retorno:**

Numérico	0 quando o objeto existe ou o número do erro no AS/400.

☑ **Exemplo:**

```
nError := TCChkObj("CALCCUST","PRODUCAO","*PGM")
```

Importante

Função para uso apenas com o TOPConnect em servidores AS/400.

TCEXEERROR()

Retorna uma string com a mensagem de erro retornada pela execução das funções TCPGMEXE() e TCSYSEXE().

☑ **Sintaxe: TCExeError()**

☑ **Parâmetros:**

Nenhum	.

☑ **Retorno:**

String	Mensagem de erro.

☑ **Exemplo:**

```
If TCPGMEXE("CALCCUST") != 0
      ALERT(TCExeError())
Endif
```

Importante

Função para uso apenas com o TOPConnect em servidores AS/400.

TCPGMEXE()

Executa um programa no servidor AS/400.

☑ **Sintaxe: TCPGMEXE(cProgram)**

☑ **Parâmetros:**

cProgram	Nome do programa a ser executado.

☑ **Retorno:**

Numérico	0 quando o objeto existe ou o número do erro no AS/400.

☑ **Exemplo:**

```
If TCPGMEXE("CALCCUST") != 0
      ALERT(TCExeError())
Endif
```

Importante

Função para uso apenas com o TOPConnect em servidores AS/400.

TCSYSEXE()

Permite a execução de um comando utilizando a sintaxe e recursos nativos do DB2/400, diretamente no servidor do banco de dados.

☑ **Sintaxe: TCSysExe(cCommand)**

☑ **Parâmetros:**

cCommand	Comando a ser executado no servidor de banco de dados utilizando sintaxe DB2/400.

☑ **Retorno:**

Numérico	0 quando o objeto existe ou o número do erro no AS/400.

☑ **Exemplo:**

```
cCommand := "CRTCBLMOD MODULE("+cTDataBase+"/"+cName+"F"+cCrrEmp+") "
cCommand += "SRCFILE("+cTDataBase+"/QSPSRC) "
cCommand += "SRCMBR("+cName+"F"+cCrrEmp+") "
cCommand += "REPLACE(*YES)"

If TCSysExe( cCommand )!= 0
      ALERT(TCExeError())
Endif
```

Função para uso apenas com o TOPConnect em servidores AS/400.

Funções acessórias para uso em fontes com interação com o TOTVS DbAccess

CHANGEQUERY()

Função que efetua as adequações necessárias a query para que a mesma possa ser executada adequadamente no banco de dados em uso pela aplicação ERP através do TopConnect.

Esta função é necessária pois a aplicação ERP Protheus pode ser utilizada com diversos bancos de dados, e cada banco possui particularidades em sua sintaxe, de forma que mesmo uma query escrita respeitando o padrão SQL ANSI pode necessitar de adequações.

☑ **Sintaxe: ChangeQuery(cQuery)**

☑ **Parâmetros:**

cQuery	String contendo a query escrita em sintaxe SQL ANSI.

☑ **Retorno:**

String	Query adequada em função do banco em uso pela conexão ativa com a aplicação TopConnect.

☑ **Exemplo:**

```
cQuery := "SELECT DISTINCT CV8_PROC FROM "+RETSQLNAME("CV8")
cQuery += " WHERE "
cQuery += "CV8_FILIAL = '"+MV_PAR01+"' AND "
cQuery += "D_E_L_E_T_ = '' "
cQuery += "ORDER BY CV8_PROC"
cQuery := ChangeQuery(cQuery)
dbUseArea(.T., "TOPCONN", TCGenQry(,,cQuery), "CV8QRY", .F., .T.)
```

Importante

A função ChangeQuery() deve obrigatoriamente ser utilizada pelos fontes da aplicação ERP, pois a mesma pode ser instalada em diversos ambientes, utilizando os vários bancos de dados homologados, mas para desenvolvimentos específicos de clientes, o analista pode optar por não utilizar a função ChangeQuery().

A razão disto é permitir ao analista, que tenha conhecimentos mais apurados do banco de dados em uso pelo cliente, utilizar uma sintaxe focada em utilizar melhor os recursos do banco, otimizando a performance da aplicação.

RETFULLNAME()

Retorna o nome real da tabela no banco de dados para utilização desta na seleção da query.

Importante

A função RetFullName() não é exclusiva para uso em queries, sendo utilizada pela aplicação ERP para verificar o nome da tabela para o alias, especificado no SX2 da empresa indicada.

☑ **Sintaxe: RetFullName(cAlias, cEmp)**

☑ **Parâmetros:**

cAlias	Nome da área de trabalho da aplicação para identificação da tabela no banco de dados.
cEmp	Código da empresa a ser avaliada.

☑ **Retorno:**

String	Nome completo / real da tabela no banco de dados.

☑ **Exemplo:**

```
SA1->(DbSetOrder(1)) // A1_FILIAL+A1_COD+A1_LOJA
cOrder := SqlOrder(IndexKey())

cQuery := "SELECT * FROM "+RETSQLNAME("SA1")+ " WHERE "
cQuery += RetSqlCond("SA1")
cQuery += " ORDER BY "+cOrder

dbUseArea(.T.,"TOPCONN", TcGenQry(,,cQuery), "TRBSQL", .T., .F.)

cRealName := RetFullName("TRBSQL")
```

RETSQLCOND()

Retorna uma string com as condições padrões de uso em uma query.

☑ **Sintaxe: RetSqlCond(cAlias)**

☑ **Parâmetros:**

cAlias	Nome do alias para composição a string de condições.

☑ **Retorno:**

String	Condições de seleção padrões para o alias indicado. As condições padrões são: ALIAS_FILIAL = xFilial(ALIAS) .AND. ALIAS.D_E_L_E_T_ = “”.

☑ **Exemplo:**

```
cQuery := "SELECT DISTINCT CV8_PROC FROM "+RETSQLNAME("CV8")
cQuery += " WHERE "
cQuery += RetSqlCond("CV8")
cQuery += "ORDER BY CV8_PROC"
cQuery := ChangeQuery(cQuery)
dbUseArea(.T., "TOPCONN", TCGenQry(,,cQuery), "CV8QRY", .F., .T.)
```

RETSQLNAME()

Retorna o nome padrão da tabela para seleção no banco de dados através da query.

☑ **Sintaxe: RetSqlName(cAlias)**

☑ **Parâmetros:**

cAlias	Alias para ser avaliado o nome padrão da tabela.

☑ **Retorno:**

String	Nome completo da tabela para seleção através da query.

☑ **Exemplo:**

```
cQuery := "SELECT DISTINCT CV8_PROC FROM "+RETSQLNAME("CV8")
cQuery += " WHERE "
cQuery += RetSqlCond("CV8")
cQuery += "ORDER BY CV8_PROC"
cQuery := ChangeQuery(cQuery)
dbUseArea(.T., "TOPCONN", TCGenQry(,,cQuery), "CV8QRY", .F., .T.)
```

RETSQLTABLE()

Retorna o nome real da tabela para seleção no banco de dados através da query.

☑ **Sintaxe: RetSqlTable(cAlias)**

☑ **Parâmetros:**

cAlias	Alias para ser avaliado o nome real da tabela.

☑ **Retorno:**

String	Nome real da tabela no banco de dados.

☑ **Exemplo:**

```
cQuery := "SELECT DISTINCT CV8_PROC FROM "+RETSQLTABLE("CV8")
cQuery += " WHERE "
cQuery += RetSqlCond("CV8")
cQuery += "ORDER BY CV8_PROC"
cQuery := ChangeQuery(cQuery)
dbUseArea(.T., "TOPCONN", TCGenQry(,,cQuery), "CV8QRY", .F., .T.)
```

SQLCOPY()

Cria um arquivo no formato especificado pela configuração LOCALFILES do ambiente com o retorno da query.

☑ **Sintaxe: SqlCopy(cFile, cWhere, aStru, cAlias, aDates, lRecno)**

☑ **Parâmetros:**

cFile	Nome do arquivo temporario destino a ser gerado
cWhere	Condicao "Where" da query
aStru	Array com os Campos a serem selecionados pela query
cAlias	Alias origem dos dados
aDates	Array com os nomes dos campos que deverão ter o tratamento da função TCSetField() para o formato de data.
lRecno	Se copia o conteúdo da coluna R_E_C_N_O_ para o arquivo gerado.

☑ **Retorno:**

Nenhum	.

SQLORDER()

Converte a sintaxe de um indíce em formato ISAM (CODEBASE) para o formato SQL ANSI

☑ **Sintaxe: SqlOrdem(cOrder)**

☑ **Parâmetros:**

cOrder	Sintaxe de índice em formato ISAM.

☑ **Retorno:**

String	Índice convertido para sintaxe SQL ANSI

☑ **Exemplo:**

```
SA1->(DbSetOrder(1)) // A1_FILIAL+A1_COD+A1_LOJA
cOrder := SqlOrder(IndexKey())

cQuery := "SELECT * FROM "+RETSQLNAME("SA1")+ " WHERE "
cQuery += RetSqlCond("SA1")
cQuery += " ORDER BY "+cOrder

dbUseArea(.T.,"TOPCONN", TcGenQry(,,cQuery), "TRBSQL", .T., .F.)

cRealName := RetFullName("TRBSQL")
```

Anotações

SQLTOTRB()

Preenche um arquivo temporário com o conteúdo do retorno da query.

☑ **Sintaxe: SqlToTrb(cQuery, aStru, cAlias)**

☑ **Parâmetros:**

cQuery	Query para seleção dos registros no banco de dados.
aStru	Array contendo a estrutura do arquivo temporário a ser preenchido com os dados de retorno da query.
cAlias	Alias do arquivo temporário a ser preenchido com os dados de retorno da query.

Para correta utilização da função SQLTOTRB() o arquivo temporário deve estar criado e com a área de trabalho (ALIAS) definido.

☑ **Retorno:**

Nenhum	.

☑ **Exemplo:**

```
// Arquivo de trabalho que será utilizado pela MaWndBrowse

cCampos:= ""
nX     := 0
nMax   := Len(aStruSQL)

aEval( aStruSQL,{|aCampo| nX++, cCampos += aCampo[1] +;
       IIF(nX == nMax,' ',', ')})

cArqTrb := CriaTrab(aStruSQL,.T.)
dbUseArea(.T.,__LOCALDRIVER,cArqTrb,cAlias,.T.,.F.)
cQuery := "SELECT "+cCampos+" FROM "+RetSqlName("SA1")+" (NOLOCK) "
SqlToTrb(cQuery,aStruSQL,cAlias)

cArqInd := CriaTrab(Nil,.F.)
cChave      := "A1_FILIAL+A1_COD+A1_LOJA"
IndRegua(cAlias,cArqInd,cChave,,,"Indexando Registros...")

dbSelectArea( cAlias )
dbGotop()
```

40.3. Aplicações com controle de comunição com o Banco de Dados

Utilizando as função de interação com a aplicação TopConnect / TOTVS DbAccess têm-se um grande número de aplicações que podem ser desenvolvidas, combinando-se recursos previamente estudados nos módulos Básico, Intermediário e utilizando os conceitos de orientação à objetos e as classes de objetos disponíveis na linguagem ADVPL, tais como:

- ☑ Rotinas de processamento utilizando queries para seleção dos registros;
- ☑ Relatórios para seleção dos registros;
- ☑ Rotinas de atualização para execução dos comando de atualização diretamente no banco de dados.

Em resumo, qualquer aplicação do ambiente ERP pode ser escrita utilizando os recursos de interação com a aplicação TopConnect / TOTVS DbAccess, para obter, manipular e atualizar os dados do banco do banco de dados.

Neste tópico serão detalhadas duas aplicações utilizando estas funções, cujo grau de complexidade ilustra bem o potencial desta ferramenta.

- ☑ **MaWndBrowse com Alias Temporário gerado por Query**

- ☑ **Banco de dados de interface**

40.3.1. MaWndBrowse com Alias Temporário gerado por Query

Conforme descrito no tópico 6.1. MaWndBrowse, esta função permite a montagem de um browse cuja estrutura do arquivo não necessita estar definida no dicionário de dados da aplicação ERP.

Utilizando esta particularidade da MaWndBrowse é possível então compor um alias temporário contendo o retorno de uma query e exibi-lo em um browse.

A vantagem deste “BrowseSQL” em relação ao browse de uma tabela “normal” é a possibilidade de combinar informações de várias tabelas através de uma query e exibi-las em tela.

Algumas aplicações interessantes para este recurso seriam:

- ☑ Consulta Kardex em tela, pois este relatório combina dados das tabelas de itens dos documentos de entrada (SD1), itens dos documentos de saída (SD2) e movimentações internas (SD3).

- ☑ Conciliação de contabilizações, pois uma contabilização é composta por uma origem, a qual pode ser praticamente de qualquer tabela de movimentos da aplicação ERP, e de um ou mais lançamentos contábeis (CT2) gerados por esta origem.

- ☑ Demonstração em tela das depreciações dos bens do imobilizado por período, compondo dinamicamente as colunas de depreciação de acordo com o intervalo de meses selecionados. Este recurso é interessante pois elimina a limitação de espaço da impressão e pode ser combinado com a funcionalidade de exportação de grids de dados para o Microsoft Excel.

Exemplo: MaWndBrowse com Alias Temporário gerado por Query

Este exemplo de uso da MaWndBrowse com alias de temporário utilizando queries é derivado do exemplo de utilização da MaWndBrowse com arquivo temporário descrito no item 6.1. MaWndBrowse.

```
#include "protheus.ch"

/*/
+-----------------------------------------------------------------------
| Função    | WndSQLTRB     | Autor | Arnaldo R. Junior | Data |      |
+-----------------------------------------------------------------------
| Descrição | Demonstra a utilização da MaWndBrowse com SQL           |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                              |
+-----------------------------------------------------------------------
/*/


User Function WndSQLTRB()

// Variáveis para o Arquivo Temporario
Local cChave      := ""
Local cArqTrb     := ""
Local aStruTRB  :=  {}
Local aStruSQL := SA1->(DbStruct())

// Variáveis para o MaWndBrowse
Local cTitulo   := "Cadastro Temporario"// Título obrigatório
Local cAlias    := "SA1" // Alias da tabela corrente podendo ser TRB
Local cFunLeg   := "" // Função que deverá retornar um valor lógico e com isso
será atribuído semaféro na primeira coluna do browse
Local cTopFun   := "" // Mostrar os registros com a chave de
Local cBotFun   := "" // Mostrar os registros com a chave ate
Local lCentered := .T. // Valor verdadeiro centraliza
Local aResource := {} // aAdd(aResource,{"IMAGEM","Texto significativo"})
Local nModelo   := 1 // 1- Menu do aRotina
Local aPesqui   := {} // aAdd(aPesqui{"Título",nOrdem}), se não passado será
utilizado o AxPesqui
Local cSeek     := "" // Chave principal para a busca, exemplo: xFilial("???")
Local lDic      := .T. // Parâmetro em conjunto com aCampos
Local lSavOrd   := .T. // Estabelecer a ordem após pesquisas.

// Variaveis para a MsAdvSize
Local lEnchBar  := .F. // Se a janela de diálogo possuirá enchoicebar (.T.)
Local lPadrao   := .F. // Se a janela deve respeitar as medidas padrões do
Protheus (.T.) ou usar o máximo disponível (.F.)
Local nMinY     := 400 // Altura mínima da janela

Local aSize       := MsAdvSize(lEnchBar, lPadrao, nMinY)

Private cCadastro := " "
Private aCampos     := {} // Se lDic=.T. utilizará o SX3, do contrário o aCampos
informado -> aAdd(aCampo,{X3_CAMPO,X3_PICTURE,X3_TITULO,X3_TAMANHO})
Private aRotina   := {} // Idêntico ao aRotina para mBrowse
```

Continuação:

```
aAdd(aRotina,{"Visualizar","U_TcVisual",0,2}) // Desenvolver Enchoice para
campos de arquivo temporário
aAdd(aRotina,{"Incluir"   ,"U_TcInclui",0,3}) // Desenvolver Enchoice para
campos de arquivo temporário
aAdd(aRotina,{"Alterar"   ,"U_TcAltera",0,4}) // Desenvolver Enchoice para
campos de arquivo temporário


aAdd(aRotina,{"Excluir"   ,"U_TcExclui",0,5}) // Desenvolver Enchoice para
campos de arquivo temporário

// Estrutura do Arquivo: Nome do campo / tipo, tamanho, decimais (SX3 para
temporário)
/*
AADD(aStruTRB,{"TRB_FILIAL"         ,"C",02,0})
// Nome_Campo , Tipo_Campo, Tamanho, Decimal
AADD(aStruTRB,{"TRB_ID"             ,"C",14,0})
AADD(aStruTRB,{"TRB_NOME"           ,"C",20,0})
AADD(aStruTRB,{"TRB_IDADE"          ,"N",03,0})
AADD(aStruTRB,{"TRB_STATUS"         ,"C",01,0})
*/
// aCampos padrão para a MaWndBrowse
//AADD(aCampos,{<Nome_Campo>,<Picture>,<Titulo>,<Tamanho>})

// aCampos melhorado para a WndBrwTRB
//AADD(aCampos,{<Nome_Campo>,<Picture>,<Titulo>,<Tamanho>,<Tipo>,<cWhen>,;
//<lObrigatorio>})
// Nota: lObrigatorio deve ser sempre a ultima informacao do aCampos
/*
AADD(aCampos,{"TRB_FILIAL"          ,"@!" ,"Filial"         ,02,"C",".F.",.T.})
AADD(aCampos,{"TRB_ID"              ,"@!" ,"Matricula"      ,14,"C",".F.",.T.})
AADD(aCampos,{"TRB_NOME"            ,"@!" ,"Nome"           ,20,"C",".T.",.F.})
AADD(aCampos,{"TRB_IDADE"           ,"@E 999","Idade"       ,03,"N",".T.",.F.})
AADD(aCampos,{"TRB_STATUS"          ,"@!" ,"Status"         ,01,"C",".T.",.T.})
*/

If ( Select( cAlias ) <> 0 )
      dbSelectArea ( cAlias )
      dbCloseArea ()
Endif

// Arquivo de trabalho que será utilizado pela MaWndBrowse

cCampos:= ""
nX     := 0
nMax   := Len(aStruSQL)
aEval( aStruSQL,{|aCampo| nX++, cCampos += aCampo[1] +;
       IIF(nX == nMax,' ',', ')})

cArqTrb := CriaTrab(aStruSQL,.T.)
dbUseArea(.T.,__LOCALDRIVER,cArqTrb,cAlias,.T.,.F.)
cQuery := "SELECT "+cCampos+" FROM "+RetSqlName("SA1")+" (NOLOCK) "
SqlToTrb(cQuery,aStruSQL,cAlias)
```

Continuação:

```
cArqInd := CriaTrab(Nil,.F.)
cChave      := "A1_FILIAL+A1_COD+A1_LOJA"

IndRegua(cAlias,cArqInd,cChave,,,"Indexando Registros...")

dbSelectArea( cAlias )
dbGotop()

MaWndBrowse(aSize[7],aSize[2],aSize[6],aSize[5],cTitulo,cAlias,/*aCampos*/,;
            aRotina,,cTopFun,cBotFun,lCentered,,nModelo,,cSeek,lDic,lSavOrd)


If ( Select( cAlias ) <> 0 )
      dbSelectArea ( cAlias )
      dbCloseArea ()
Endif

If File(cArqInd+OrdBagExt())
      FErase(cArqInd+OrdBagExt())
ENDIF

DbSelectArea("SA1")
DbSetOrder(1)

Return

/*/
+-----------------------------------------------------------------------
| Função    | TcVisual      | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | Enchoice para arquivos temporarios                        |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/


USER FUNCTION TcVisual(cAlias,nReg,nOpc)

LOCAL aCposEnch := {}
LOCAL nLinha      := 15
LOCAL nColuna     := 10
LOCAL nOpcE       := aRotina[nOpc][4] // Opcao de verdade
LOCAL bOk         := {||oDlg:End()}
LOCAL bCancel     := {||oDlg:End()}
LOCAL nX

// Padrao Modelo2(), com lWhen -> cWhen
// AADD(aCposEnch,{"<Variavel>"     ,{nLinha,nColuna}
// ,"<Titulo>","<Picture>","<Validacao>","<F3>","<lWhen>",<Tamanho>})
// aCampos,{"TRB_FILIAL"            ,"@!" ,"Filial"         ,02}
```

Continuação:

```
For nX := 1 to Len(aCampos)

      If nX > 1
            nLinha  := IIF(nX%2 > 0 ,nLinha  := nLinha +015,nLinha) // Impar
            nColuna := IIF(nX%2 == 0,nColuna := nColuna+170,10)// Par
      Endif

      AADD(aCposEnch,{"_"+aCampos[nX][1],{nLinha,nColuna} ,;
                           aCampos[nX][3],aCampos[nX][2],"AllwaysTrue()",;
                           "",".F.",aCampos[nX][4]})

      SetPrvt("_"+aCampos[nX][1])

      &("_"+aCampos[nX][1]) := (cAlias)->&(aCampos[nX][1])

Next nX

oDlg := TDialog():New(000,000,400,650,cCadastro,,,,,,,,,.T.)

TEnchoice(oDlg, aCposEnch,,,,,.T.)

oDlg:bInit := {|| EnchoiceBar(oDlg, bOk, bCancel,.F.,{},nReg,cAlias)}
oDlg:lCentered := .T.
oDlg:Activate()

RETURN

/*/
+-----------------------------------------------------------------------
| Função    | TcInclui      | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | Enchoice para arquivos temporarios                        |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

USER FUNCTION TcInclui(cAlias,nReg,nOpc)

LOCAL aCposEnch := {}
LOCAL nLinha      := 15
LOCAL nColuna     := 10
LOCAL nOpcE       := aRotina[nOpc][4] // Opcao de verdade
LOCAL bOk
LOCAL bCancel     := {||oDlg:End()}
LOCAL aArea       := GetArea()
LOCAL nX

// Padrao Modelo2(), com lWhen -> cWhen
// AADD(aCposEnch,{"<Variavel>"     ,{nLinha,nColuna}
//,"<Titulo>","<Picture>","<Validacao>","<F3>","<lWhen>",<Tamanho>})
// aCampos,{"TRB_FILIAL"            ,"@!" ,"Filial"         ,02}
```

Continuação:

```
For nX := 1 to Len(aCampos)

      If nX > 1
            nLinha  := IIF(nX%2 > 0 ,nLinha  := nLinha +015,nLinha) // Impar
            nColuna := IIF(nX%2 == 0,nColuna := nColuna+170,10)// Par
      Endif

      AADD(aCposEnch,{"_"+aCampos[nX][1],{nLinha,nColuna} ,;
                           aCampos[nX][3],aCampos[nX][2],"AllwaysTrue()",;
                           "",".T.",aCampos[nX][4]})

      SetPrvt("_"+aCampos[nX][1])

      Do Case
            Case aCampos[nX][5] == "C"
                  &("_"+aCampos[nX][1]) := Space(aCampos[nX][4])
            Case aCampos[nX][5] == "N"
                  &("_"+aCampos[nX][1]) := 0
            Case aCampos[nX][5] == "D"
                  &("_"+aCampos[nX][1]) := CTOD("")
            Case aCampos[nX][5] == "L"
                  &("_"+aCampos[nX][1]) := .F.
            Case aCampos[nX][5] == "M"
                  &("_"+aCampos[nX][1]) := Space(aCampos[nX][4])
      EndCase

Next nX

oDlg := TDialog():New(000,000,400,650,cCadastro,,,,,,,,,.T.)

TEnchoice(oDlg, aCposEnch,,,,,.T.)

bOk := {|| IIF( U_TValid(cAlias,nReg,nOpcE,aCampos),;
              ( U_TGravar(cAlias,nReg,nOpcE,aCampos),oDlg:End()),)}

oDlg:bInit := {|| EnchoiceBar(oDlg, bOk, bCancel,.F.,{},nReg,cAlias)}
oDlg:lCentered := .T.
oDlg:Activate()

RETURN
```

Anotações

Continuação:

```
/*/
+-----------------------------------------------------------------------
| Função    | TcAltera      | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | Enchoice para arquivos temporarios                        |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

USER FUNCTION TcAltera(cAlias,nReg,nOpc)

LOCAL aCposEnch := {}
LOCAL nLinha      := 15
LOCAL nColuna     := 10
LOCAL nOpcE       := aRotina[nOpc][4] // Opcao de verdade
LOCAL bOk
LOCAL bCancel     := {||oDlg:End()}
LOCAL aArea       := GetArea()
LOCAL nX

// Padrao Modelo2(), com lWhen -> cWhen
// AADD(aCposEnch,{"<Variavel>"     ,{nLinha,nColuna}
//,"<Titulo>","<Picture>","<Validacao>","<F3>","<lWhen>",<Tamanho>})
// aCampos,{"TRB_FILIAL"            ,"@!" ,"Filial"         ,02}

For nX := 1 to Len(aCampos)

      If nX > 1
            nLinha  := IIF(nX%2 > 0 ,nLinha  := nLinha +015,nLinha) // Impar
            nColuna := IIF(nX%2 == 0,nColuna := nColuna+170,10)// Par
      Endif

      AADD(aCposEnch,{"_"+aCampos[nX][1],{nLinha,nColuna} ,;
                      aCampos[nX][3],aCampos[nX][2],"AllwaysTrue()",;
                      "",aCampos[nX][6],aCampos[nX][4]})

      SetPrvt("_"+aCampos[nX][1])

      &("_"+aCampos[nX][1]) := (cAlias)->&(aCampos[nX][1])

Next nX

oDlg := TDialog():New(000,000,400,650,cCadastro,,,,,,,,,.T.)

TEnchoice(oDlg, aCposEnch,,,,,.T.)

bOk := {|| IIF( U_TValid(cAlias,nReg,nOpcE,aCampos),;
              ( U_TGravar(cAlias,nReg,nOpcE,aCampos),oDlg:End()),)}

oDlg:bInit := {|| EnchoiceBar(oDlg, bOk, bCancel,.F.,{},nReg,cAlias)}
oDlg:lCentered := .T.
oDlg:Activate()

RETURN
```

Continuação:

```
/*/
+-----------------------------------------------------------------------
| Função    | TcExclui      | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | Enchoice para arquivos temporarios                        |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

USER FUNCTION TcExclui(cAlias, nReg, nOpc)

LOCAL aCposEnch := {}
LOCAL nLinha      := 15
LOCAL nColuna     := 10
LOCAL nOpcE       := aRotina[nOpc][4] // Opcao de verdade
LOCAL bOk
LOCAL bCancel     := {||oDlg:End()}
LOCAL nX

// Padrao Modelo2(), com lWhen -> cWhen
// AADD(aCposEnch,{"<Variavel>"     ,{nLinha,nColuna}
// ,"<Titulo>","<Picture>","<Validacao>","<F3>","<lWhen>",<Tamanho>})
// aCampos,{"TRB_FILIAL"            ,"@!" ,"Filial"         ,02}

For nX := 1 to Len(aCampos)

      If nX > 1
            nLinha  := IIF(nX%2 > 0 ,nLinha  := nLinha +015,nLinha) // Impar
            nColuna := IIF(nX%2 == 0,nColuna := nColuna+170,10)// Par
      Endif

      AADD(aCposEnch,{"_"+aCampos[nX][1],{nLinha,nColuna} ,;
                        aCampos[nX][3],aCampos[nX][2],"AllwaysTrue()",;
                        "",".F.",aCampos[nX][4]})

      SetPrvt("_"+aCampos[nX][1])

      &("_"+aCampos[nX][1]) := (cAlias)->&(aCampos[nX][1])

Next nX

oDlg := TDialog():New(000,000,400,650,cCadastro,,,,,,,,,.T.)

TEnchoice(oDlg, aCposEnch,,,,,.T.)

bOk := {|| IIF( U_TValid(cAlias,nReg,nOpcE,aCampos),;
                        (U_TGravar(cAlias,nReg,nOpcE,aCampos),oDlg:End()),)}

oDlg:bInit := {|| EnchoiceBar(oDlg, bOk, bCancel,.F.,{},nReg,cAlias)}
oDlg:lCentered := .T.
oDlg:Activate()

RETURN
```

Continuação:

```
/*/
+-----------------------------------------------------------------------
| Função    | TcValid       | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | Enchoice para arquivos temporarios                        |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

USER FUNCTION TcValid(cAlias,nReg,nOpc,aCampos)
LOCAL lRet
LOCAL nX
LOCAL nPosObrig := Len(aCampos[1])

For nX := 1 to Len(aCampos)
      IF aCampos[nX,nPosObrig] == .T.
            IF !(lRet := !Empty(&("_"+aCampos[nX,1])))
                  Help("TEnchoice",1,"HELP","OBRIGATORIO","Existem campos
obrigatorios nao preenchidos",1,0)
                  RETURN lRet // EXIT
            ENDIF
      ENDIF
Next nX

IF nOpc == 3
      IF !(lRet := !((cAlias)->(dbSeek(_TRB_FILIAL+_TRB_ID))))
            Help("TEnchoice",1,"HELP","INCLUSAO","Ja existe um registro com esta
chave",1,0)
      ENDIF
ELSE
      IF !(lRet := (cAlias)->(dbSeek(_TRB_FILIAL+_TRB_ID)))
            Help("TEnchoice",1,"HELP","ALTERACAO","Nao existe um registro com
esta chave",1,0)
      ENDIF
ENDIF

RETURN lRet
```

Anotações

Continuação:

```
/*/
+-----------------------------------------------------------------------
| Função    | TcGravar      | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | Enchoice para arquivos temporarios                        |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/


USER FUNCTION TcGravar(cAlias,nReg,nOpc,aCampos)
LOCAL nX

RecLock(cAlias,nOpc==3)
      IF nOpc == 5
            DbDelete()
      ELSE
            For nX := 1 to Len(aCampos)
                  (cAlias)->&(aCampos[nX][1]) := &("_"+aCampos[nX][1])
            Next nX
      ENDIF
Msunlock()

RETURN
```

Anotações

Continuação:

```
/*/
+-----------------------------------------------------------------------
| Função    | TEnchoice     | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | Enchoice para arquivos temporarios                        |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

Static Function TEnchoice(oDlg, aCampos, nLeftE, nTopE, nHeightE, nWidthE,;
lEnchBar)

Local aSays       := {}
Local aGets       := {}
Local cCaption    := ""
Local cPict       := ""
Local cValid      := ""
Local cF3         := ""
Local cWhen       := ""
Local cBlKSay     := ""
Local cBlkGet     := ""
Local cBlKVld     := ""
Local cBlKWhen    := ""
Local nLeft       := 0
Local nTop        := 0
Local nI          := 0

Default lEnchBar  := .F.
Default     nLeftE            := IIF(lEnchBar,(oDlg:nLeft)+16,2)
Default     nTopE       := 2
Default     nHeightE    := (oDlg:nHeight)-135
Default nWidthE         := (oDlg:nWidth)-102
// Padrao Modelo2(), com lWhen -> cWhen
// AADD(aC,{"<Variavel>"      ,{nLinha,nColuna}
//,"<Titulo>","<Picture>","<Validacao>","<F3>","<lWhen>",<Tamanho>})

If Len(aCampos) > 0

    oScroll := TScrollBox():New(oDlg, nLeftE, nTopE, nHeightE , nWidthE, .T.,;
                                .T., .T.)

      For nI:=1 to Len(aCampos)

            If Len(aCampos[nI])==8

                  cCampo      := aCampos[nI,1]
                  nLeft := aCampos[nI,2,1]-13
                  nTop  := aCampos[nI,2,2]
                  cCaption:= Iif(Empty(aCampos[nI,3]), " "  ,aCampos[nI,3])
                  cPict := Iif(Empty(aCampos[nI,4]), Nil    ,aCampos[nI,4])
                  cValid      := Iif(Empty(aCampos[nI,5]), ".t."  ,aCampos[nI,5])
                  cF3         := Iif(Empty(aCampos[nI,6]), NIL    ,aCampos[nI,6])

                  cWhen := Iif(Empty(aCampos[nI,7]), ".T."  ,aCampos[nI,7])
```

Continuação:

```
                    nWidthG     := Iif(Empty(aCampos[nI,8]), 100,;
                                   IiF(aCampos[nI,8]*3.5 > 100,100,nil))

                    cBlKSay := "{|| OemToAnsi('"+cCaption+"')}"
                    cBlkGet := "{ | u | If( PCount() == 0,;
                                "+cCampo+","+cCampo+":= u ) }"
                    cBlKVld := "{|| "+cValid+"}"
                    cBlKWhen := "{|| "+cWhen+"}"

                    AADD(aSays,Array(1))
                    aSays[nI] := TSay():New(nLeft+1, nTop, &(cBlkSay), oScroll,,,;
                                 .F., .F., .F., .T.,,, 50, 8, .F., .F., .F., .F.,;
                                 .F. )
                    AADD(aGets,Array(1))

                    aGets[nI] := TGet():New( nLeft, nTop+50, &(cBlKGet), oScroll,;
                                  nWidthG, 11, cPict, &(cBlkVld),,,, .F.,, .T.,,;
                                  .F., &(cBlkWhen), .F., .F.,, .F., .F. , cF3,;
                                  (cCampo))

            EndIf
      Next
Endif

Return
```

Anotações

40.3.2. Banco de dados de interface

O termo “Banco de dados de interface” é utilizado para classificar uma aplicação ADVPL escrita com funções de interação com a aplicação TopConnect / TOTVS DbAccess para realizar a integração entre a aplicação ERP Protheus e outro sistema, utilizando como meio de comunicação um banco de dados e não mais importação de arquivos em “N” formatos, tais como .TXT, .DBF e similares.

Este tipo de integração não visa substituir formatos de integrações padrões já existentes como o padrão CNAB e o padrão Web XML.
Sua aplicação é integração direta entre sistemas através de um banco de dados com formato e padrões comuns entre elas.

As vantagens da utilização do conceito de Banco de Dados de Interface em comparação da integração por importação / exportação de arquivos são:

- ☑ Maior controle da integração das informações entre os sistemas, permitindo um tratamento adequado de eventos de erros;
- ☑ Transparência e rastreabilidade das informações integradas, implementando confiabilidade no processo;
- ☑ Possibilidade de utilização de um tipo de banco de dados pela aplicação ERP Protheus e de outro tipo de banco de dados para integração entre os sistemas.
- ☑ Possibilidade de desenvolvimento de ferramentas para monitorar e gerenciar a integração entre os sistemas através do controle da área de interface no banco de dados, inclusive com a utilização de Stored Procedures para aumentar a performance da aplicação de interface.

Considerações relevantes sobre as funções TCLink() e TCSetConn()

- ☑ A função TCLink() permite a abertura de uma conexão com um banco de dados, desde que haja o OBDC configurado no ambiente da aplicação e na aplicação TOPConnect / TOTVS DBAccess;
- ☑ Cada abertura de conexão através da função TcLink() consome uma licensa de usuário do TOPConnect / TOTVS DBAccess. Após a abertura de uma conexão a mesma deve ser selecionada pela função TcSetConn() para se tornar a conexão ativa;
- ☑ Podem ser utilizadas diversas conexões em uma mesma aplicação ADVPL, e podem ser utilizadas as informações das tabelas já abertas e vinculadas a uma área de trabalho independente da conexão que está ativa. Somente deve ser verificado que para abertura de uma tabela e vínculo desta com uma área de trabalho obrigatoriamente deve estar ativa a conexão com o banco de dados que contém esta tabela.
- ☑ A conexão aberta entre o ERP Protheus e o TopConnect / DbAccess possuí o número de referência 0. Desta forma, após finalizar uma conexão aberta com o comando TCLink() deve ser efetuado um TCSetConn(0) para restaurar como conexão ativa a conexão padrão do Protheus com o TopConnec.

A figura abaixo ilustra a estrutura típica de uma aplicação utilizando o conceito de Banco de Dados de Interface, com foco na aplicação ERP Protheus e seus recursos pré-disponíveis:

Tarefa	Descrição	Detalhamento
01	**Exportação Protheus -> DbInterface**	Utilização de queries ou stored procedures para atualização das tabelas do banco DBInterface no formato necessário ao ERP X, executadas a partir do ERP Protheus.
02	**Importação DbInterface -> ERP X**	Ferramentas proprietárias do ERP X para leitura do DBInterface e tratamento das informações para atualização do banco de dados desta aplicação.
03	**Extração ERP X -> DbInterface**	Pode ser uma ferramenta proprietária da aplicação ERP X, ou pode ser um conjunto de queries e stored procedures desenvolvidas em conjunto com a equipe do ERP X e a equipe TOTVS, permitindo o controle de execução da extração pelo ERP Protheus.
04	**Importação DbInterface -> Protheus**	Utilização de queries ou stored procedures para leitura das informações do DBInterface e atualização de tabelas intermediárias no banco de dados do ERP Protheus, executadas a partir do ERP Protheus.
05	**Efetivação Protheus -> Protheus**	Utilização de funções próprias para esta finalidade no ERP Protheus para efetivação das informações importadas em informações válidas no ERP. Estas funções normalmente são compatíveis com a execução através do recurso de MsExecAuto().

Considerações complementares sobre o conceito de Banco de Dados de Interface

- ☑ A utilização de tabelas intermediárias no banco de dados Protheus apesar de não ser obrigatória é recomendada para permitir um tratamento adequado das ocorrências de efetivação das informações, principalmente quando utilizada a função MsExecAuto() para executação das rotinas de efetivação dos dados;

- ☑ Para extração das informações tanto do banco de dados Protheus para exportação, quanto do banco de dados do sistema ERP X é recomendável o uso de Stored Procedures para aumentar a performance do processo como um todo. É importante lembrar que se o sistema ERP X possuir procedures de exportação as mesmas poderão ser executadas diretamente pelo Protheus através da função TCSpExec();

- ☑ É recomendável que as funcionalidades de Exportação, Extração, Importação e Efetivação sejam tratadas tanto com interfaces de execução direta quanto através de execução via Job.

Anotações

40.4. Embedded SQL – Facilitador de Query's

O objetivo do Embedded SQL é facilitar a escrita e leitura de query's. Foi definida uma sintaxe para que se possa escrever a query diretamente no código ADVPL, sem a necessidade de ficar concatenando pedaços de string para compor a string final.

Disponibilidade do Recurso

Este recurso está disponível apenas no Protheus 8. A utilização do Embedded SQL divide-se em : compilação do fonte, e execução do fonte. Para ser possível compilar um fonte com o código escrito no formato Embedded, deve ser utilizado um Protheus, com Build igual ou superior a 7.00.050721p, em um ambiente com repositório para TopConnect (RPODB=TOP).

A utiliação deste recurso também depende da atualização da LIB 811, superior a 'Lib 20050902 - 811'.

☑ **Exemplo básico - Fonte atual**

```
cQuery : 'SELECT SE2.E2_PREFIXO,SE2.E2_NUM '
cQuery += 'FROM '+RetSqlTable('SE2')+' SE2,'+RetSqlTable('QEK')+' QEK '
cQuery += 'WHERE SE2.E2_FILIAL= '+xfilial('SE2')+' AND '
cQuery += 'SE2.E2_PREFIXO<> ''+cPrefixo+'' AND '
cQuery += 'SE2.D_E_L_E_T_ = ' ' '
cQuery += 'ORDER BY '+RetSqlOrder('SE2')
dbUseArea(.T.,'TOPCONN',TcGenQry(,,cQuery),'E2TEMP',.T.,.T.)
TCSetField('E2TEMP','E2_EMISSAO','D',8,0)
```

☑ **Exemplo básico - Utilizando Embedded SQL**

```
BeginSql alias 'E2TEMP'
     column E2_EMISSAO as Date
     %noparser%
     SELECT SE2.E2_PREFIXO,SE2.E2_NUM
     FROM %table:SE2% SE2,%table:QEK% QEK
     WHERE SE2.E2_FILIAL= %xfilial:SE2% AND
           SE2.E2_PREFIXO<> %exp:cPrefixo% AND
           SE2.%notDel%
     ORDER BY %Order:SE2%
EndSql
```

Características operacionais - Sintaxe

O bloco onde será escrito o Select deve sempre ser iniciado com **'BeginSql alias'** e finalizado com **'EndSql'.**

Partes do código que devem ser substituídas aparecem entre os sinais de %. Estas expressões possuem tratamento especial em momento de execução.

Qualquer instrução colocada entre BEGINSQL... ENDSQL, que não seja uma expressão %...% , será inserida na query a ser enviada para o banco, de forma literal.

Variáveis, expressões, funções aparecem iniciando com **%exp: %**.

Em **column,** especificar campos da query que são do tipo data, lógico ou numérico (DATE, LOGIC, NUMBER). Esta linha é trocada por chamadas à função **TCSetField**.

%noparser% indica que a query não deve passar pela função '**ChangeQuery**' antes de ser enviada ao banco de dados. Caso não especificado, o default é a string da query ser passada automaticamente pela função ChangeQuery.

%table:<*alias*>% é substituída por **RetSqlName(<*alias*>)**

%notDel% é substituída por **D_E_L_E_T_= ' '**

%Order:<*alias*>% é substituída por **SqlOrder(<*alias*>->(IndexKey()))**

Há 3 opções para o %Order:

1. **%Order: <*cAlias*> %** traduzido para SqlOrder(<*cAlias*>->(IndexKey()))
2. **%Order: <*cAlias*>, <*nIndice*>%** traduzido para SqlOrder(<*cAlias*>->(IndexKey(<*nIndice*>)))
3. **%Order: <*cAlias*>, <*cNick*>%** traduzido para SqlOrder(<*alias*>->(DBNickIndexKey(<*cNick*>)))

Limitação:

Não é permitido incluir funções no meio do código 'embedded'. Se precisar, o valor deve ser guardado em uma variável antes do início do BeginSql.

```
tam_cp := GetE2ValorSize()

BeginSql alias 'E2TEMP'
column E2_EMISSAO as Date, E2_VALOR as Numeric(tam_cp,2)
. . .
EndSql
```

Erros de Compilação

Caso seja utilizado algum argumento inválido para especificar as colunas, ou erros de sintaxe nas expressões a serem transformadas para a montagem da query, a compilação do fonte é interrompida com a ocorrência 'Syntax Error', informando a linha onde a primeira ocorrência foi encontrada.

```
ENDSQL (Error C2001  Syntax error:)
```

Caso a ocorrência de compilação aponte diretamente para a linha do código-fonte, onde está escrita a instrução EndSql, verifique se existe algum espaço em branco ou tabulação, a partir do início da linha, antes da instrução EndSql. A versão atual desde ambiente não suporta esta declaração, exigindo que a instrução EndSql seja alinhada à esquerda do fonte, sem espaços ou tabulações.

Erros de Execução

☑ **'Query Argument Error : Alias [XXX] already in use.'**

Caso a instrução BeginSQL especifique um alias que já esteja aberto (em uso), a aplicação é abortada com a ocorrência de erro fatal acima, informando em XXX o alias utilizado.

☑ **'Query Argument Error : Invalid Value Type [X]'**

Caso alguma expressão informada na Query, através da tag %exp: ... %, retorne um valor de tipo diferente de 'C' Catacter, 'D' Data, 'N' Numérico, ou 'L' Lógico, a aplicação é abortada com a ocorrência de erro acima, onde o tipo do argumento inesperado é mostrado em [X].

☑ **'Type Mismach on +'**

Esta ocorrência, se reproduzida, informará na pilha de chamadas o número da linha do código-fonte correspondente à instrução EndSQL. Ocorre caso alguma função intermediária do engine do Embedded SQL, excluindo-se as funções especificadas na query com a sintaxe %exp: ... % , retornar um conteúdo não-caractere a ser acrescentado na Query. Esta ocorrência é de mais difícil localização, sendo útil nestes casos a análise do arquivo temporário gerado pelo Protheus IDE, no momento da compilação.

☑ **Help NOFUNCW - Função __EXECSQL**

Caso um fonte com o Embedded SQL seja executado em um repositório que não tenha sido atualizado, OU que não seja um Repositório para o ambiente TOPConnect (RPODB=TOP), a aplicação exibirá a ocorrência acima, indicando que a função interna de execução da Query não está presente no ambiente. Verifique se a lib está atualizada, e se o RPO em uso é de um ambiente TOPConnect.

Características operacionais - depuração

Dada a montagem da Query, não é possível depurar o bloco do código-fonte compreendido entre as instruções BeginSql e EndSql, não sendo considerados pontos de parada de depuração (*BreakPoints*), caso colocados neste intervalo do código. A colocação de pontos de parada deve ser realizada antes ou depois deste bloco.

Função auxiliar - GETLastQuery()

Após a abertura do cursor, no alias especificado, a função GetLastQuery() retorna um array, com 5 elementos, onde estão disponíveis as seguintes informações sobre a Query executada :

[1] cAlias - Alias usado para abrir o Cursor.
[2] cQuery - Query executada.
[3] aCampos - Array de campos com critério de conversão especificados.
[4] lNoParser - Caso .T., não foi utilizada ChangeQuery() na String original.
[5] nTimeSpend - Tempo, em segundos, utilizado para a abertura do Cursor.

☑ **Exemplo mais completo: Código ADVPL**

```
BeginSql alias 'E2TEMP'
     column E2_EMISSAO as Date, E2_VALOR as Numeric(tam_cp,2)
     column  QEK_SKLDOC As Logical

     %noparser%

     SELECT SE2.E2_PREFIXO,SE2.E2_NUM, SE2.E2_FORNECE, SE2.E2_LOJA,SE2.E2_VALOR,
     SE2.D_E_L_E_T_ DEL1, QEK.D_E_L_E_T_ DEL2 , QEK.QEK_SKLDOC, SE2.R_E_C_N_O_
     SE2RECNO
     FROM %table:SE2% SE2,%table:qeK% QEK
     WHERE SE2.E2_FILIAL= %xfilial:SE2% AND
                    qek.%notDel%  and
                    SE2.E2_PREFIXO<> %exp:cPrefixo% AND
                    SE2.E2_NUM<> %exp:(cAlias)->M0_CODIGO% AND
                    SE2.E2_NUM<>45        AND
                    SE2.E2_FORNECE=%exp:Space(Len(SE2->E2_FORNECE))% AND
                    SE2.E2_EMISSAO<>%exp:MV_PAR06% AND
                    SE2.E2_LOJA<>%exp:MV_PAR05% AND
                    SE2.E2_VALOR<>%exp:MV_PAR04% AND
                    qek.QEK_SKLDOC<>%exp:MV_PAR03%   And
                    SE2.%notDel%
     ORDER BY %Order:SE2,1%
EndSql
```

☑ **Exemplo mais completo: Fonte gerado pelo pré-compilador (PPO)**

```
__execSql(
   'E2TEMP',
   ' SELECT SE2.E2_PREFIXO,SE2.E2_NUM, SE2.E2_FORNECE, SE2.E2_LOJA,SE2.E2_VALOR,
   SE2.D_E_L_E_T_ DEL1, QEK.D_E_L_E_T_ DEL2 , QEK.QEK_SKLDOC, SE2.R_E_C_N_O_
   SE2RECNO FROM  '+RetSqlName('SE2')+' SE2, '+RetSqlName('QEK')+' QEK WHERE
   SE2.E2_FILIAL=  '' +xFilial('SE2')+''  AND qek.D_E_L_E_T_= ' ' AND
   SE2.E2_PREFIXO<>  '+___SQLGetValue(CPREFIXO)+' AND SE2.E2_NUM <> '+
   ___SQLGetValue((CALIAS)->M0_CODIGO)+' AND SE2.E2_NUM<>45 AND SE2.E2_FORNECE= '+
   ___SQLGetValue(SPACE(LEN(SE2->E2_FORNECE)))+' AND SE2.E2_EMISSAO<> '+
   ___SQLGetValue(MV_PAR06)+' AND SE2.E2_LOJA<> '+___SQLGetValue(MV_PAR05) +' AND
   SE2.E2_VALOR<> '+___SQLGetValue(MV_PAR04)+' AND qek.QEK_SKLDOC<> '+
   ___SQLGetValue(MV_PAR03)+' And SE2.D_E_L_E_T_= ' ' ORDER BY  '+
   SqlOrder(SE2->(IndexKey(1))), {{'E2_EMISSAO','D',8,0}, {'E2_VALOR','N',tam_cp,2},
   {'QEK_SKLDOC','L',1,0}},.T.)
)
```

Anotações

41. Funcionalidade MsExecAuto

Sobre a MsExecAuto e Rotinas Automáticas

A funcionalidade MsExecAuto, ou também conhecida como Rotina Automática, permite a execução de rotinas do ambiente ERP Protheus por funções específicas, o que confere ao desenvolvedor a possibilidade de efetuar tratamentos específicos antes da execução da rotina padrão, e mais importante, não perder nenhuma funcionalidade que a rotina padrão oferece.

Avaliando esta funcionalidade apenas pelo parágrafo acima, tem-se a impressão de ser um recurso simples e até mesmo desnecessário, pois um desenvolvedor experiente poderia reproduzir todas as validações e tratamentos existentes em sua rotina, então porque ter de tratar a rotina padrão?

Para responder a esta pergunta devemos fazer as seguintes considerações:

- ☑ **A aplicação ERP está em constante evolução:**

  No momento de desenvolvimento da rotina específica era necessário apenas o tratamento de um conjunto isolado de informações, mas com o aprimoramento do ERP como um todo, agora são necessários outros conjuntos de dados, os quais não foram tratados pelo desenvolvedor naquele momento, mas que estão contemplados na rotina padrão do sistema.

  Como o desenvolvedor optou por realizar todo o tratamento de forma específica, em uma atualização do ERP este desenvolvimento precisará ser revisto ou até mesmo refeito, o que implacará em custo para o cliente ou para o desenvolvedor.

  Se o desenvolvedor tivesse optado por utilizar a rotina padrão encapsulada em seu desenvolvimento ele não teria problemas em adaptar sua rotina as novas necessidades do sistema, pois a rotina padrão já se preocupada com a compatibilidade entre as versões, possibilitando que uma simples manutenção atualize toda a rotina específica, isto se esta manutenção for necessária, pois a rotina padrão poder tratar a novas informações com "conteúdos padrões" dependendo da situação em que estas não forem informadas.

- ☑ **A aplicação ERP pode ser personalidada através de pontos de entrada e do dicionário de dados:**

  A partir do momento que uma aplicação padrão é desenvolvida e disponibilizada a mesma pode sofrer diversas personalizações em campo, as quais podem ser implementadas em paralelo ou após o desenvolvimento da rotina específica disponibilizada pelo desenvolvedor.

  Se esta situação não for prevista, a inclusão de informações utilizando a rotina padrão do sistema poderá sofrer diversas validações ou até mesmo efetuar a gravação de informações adicionais, as quais não estão visíveis na rotina específica, gerando graves inconistências na base de dados.

Avaliando estas considerações percebesse o fator de sucesso que um desenvolvimento específico pode ter ao optar por refazer todos os tratamentos do sistema em detrimento de utilizar a funcionalidade MsExecAuto, ou seja, qualquer modificação no ambiente, independente do quão simples pode tornar a rotina específica inadequada.

Quando utilizar a funcionalidade MsExecAuto ?

Tendo em vista este cenário fica a pergunta: “Quando utilizar a funcionalidade MsExecAuto ?”

A resposta pode ser óbvia e até conservadora, mas é “Sempre que a funcionalidade estiver disponível naquele processo”.

Em resumo, qualquer funcionalidade de interface, seja ela de carga ou contínua, qualquer funcionalidade de atualização, ou seja, qualquer manutenção na base de dados do sistema, sempre que possível, deve utilizar a funcionalidade MsExecAuto.

Processos da aplicação ERP com tratamentos para execução por MsExecAuto

Pode-se dizer que as principais rotinas de atualização do ERP Protheus atualmente possuem o tratamento necessário para sua execução por MsExecAuto, e com base nas necessidades de desenvolvimento internas da TOTVS e nas melhorias solicitadas por clientes e analistas de implantação as demais rotinas necessárias são atualizadas para contemplar este tratamento.

O quadro abaixo ilustra um dos últimos cenários de tratamento de MsExecAuto pelas rotinas da aplicação ERP Protheus.

Rotina	Parâmetros
ATFA010	(xAutoCab, xAutoItens, nOpcAuto)
ATFA030	(xAutoCab, nOpc)
ATFA035	(xAutoCab, nOpc)
CFGX016	(xAutoCab, xAutoItens)
CTBA015	(nOpcAuto, aAuto)
CTBA016	(nOpcAuto, aAuto)
CTBA020	(aRotAuto, nOpcAuto, aRotItem)
CTBA102	(xAutoCab, xAutoItens, nOpcAuto)
CTBA270	(xAutoCab, xAutoItens, nOpcAuto)
FATA140	(nOpcAuto, aAuto)
FATA220	(nOpcAuto, aAutoCab)
FATA300	(nOpcAuto, xAutoCab, xAutoAD2, xAutoAD3, xAutoAD4, xAutoAD9)
FATA310	(xAutoCab, xAutoItens, nOpcAuto)
FATA320	(xAutoVend, xAutoTask, xAutoCalend)
FINA010	(aRotAuto, nOpcAuto)
FINA040	(aRotAuto, nOpcAuto)
FINA050	(aRotAuto, nOpcion, nOpcAuto)
FINA070	(xAutoCab, nOpc)
FINA080	(xAutoCab, nOpc)
FINA390	(nPosArotina, xAutoCab, nOpcAuto)
MATA040	(aAuto, nOpcAuto)
MATA080	(aRotauto, nOpcAuto)
MATA103	(xAutoCab, xAutoItens, nOpcAuto, lWhenGet, xAutoImp)
MATA105	(xReservCab, xReservItens, nOpcAuto)
MATA110	(xAutoCab, xAutoItens, nOpcAuto)
MATA116A	(xAutoCab, xAutoItens, lInclui)
MATA120	(nFuncao, xAutoCab, xAutoItens, nOpcAuto,)
MATA125	(xAutoCab, xAutoItens, nOpcAuto)
MATA140	(xAutoCab, xAutoItens, nOpcAuto)
MATA145	(xAutoCab, xAutoIt, xAutoIt2, nOpcAuto)
MATA150	(xAutoCab, xAutoItens, nOpcAuto)

Rotina	Parâmetros
MATA175	(xRotAuto, xOpcAuto)
MATA185	(xAutoSCP, xAutoSD3, nOpcAuto, xPerg)
MATA200	(xAutoCab, xAutoItens, nOpcAuto)
MATA230	(aAuto, nOpcAuto)
MATA241	(xAutoCab, xAutoItens, nCallOpcx)
MATA242	(xAutoCab, xAutoItens, nOpcAuto)
MATA261	(aAutoItens, nOpcAuto)
MATA265	(xAutoCab, xAutoItens, nOpcAuto)
MATA266	(aAutoItens)
MATA267	(aAutoItens)
MATA360	(xAutoCab, xAutoItens, nOpcAuto)
MATA410	(xAutoCab, xAutoItens, nOpcAuto)
MATA415	(xAutoCab, xAutoItens, nOpcAuto)
MATA416	(xAutoCab, xAutoItens)
MATA490	(xAuto, nOpcAuto)
MATA685	(xAutoCab, xAutoItens, nOpcAuto)
MATA700	(xAuto, nOpcAuto)
MATA920	(xAutoCab, xAutoItens, nOpcAuto)
TMKA061	(xAutoCab, xAutoItens, nOpcAuto)
TMKA062	(xAutoCab, xAutoItens, nOpcAuto)
TMKA271	(xAutoCab, xAutoItens, nOpcAuto)
TMSA040	(xAutoCab, xAutoItens, nOpcAuto)
TMSA050	(xAutoCab, xAutoItens, xItensPesM3, xItensEnder, nOpcAuto)
TMSA170	(xAutoCab, nOpcAuto)
TMSA360	(xAutoCab, xAutoItens, xAutoNFAva, nOpcAuto)
TMSA430	(cAlias, xAutoCab, xAutoVei, xAutoMot, nOpcAuto)
TMSA440	(xAutoCab, xAutoItens, nOpcAuto)

Quando não utilizar a funcionalidade MsExecAuto

Apesar de todas as orientações dos tópicos anteriores existem situações nas quais não é recomendado utilizar a funcionalidade MsExecAuto, no caso, em situações para as quais existem rotinas padrões da aplicação ERP para processamentos de integrações e atualizações que possuem tratamento para execução direta ou execução via Job.

Para estes casos a rotina já está pré-disposta a execução direta, não havendo a necessidade de tratamento via MsExecAuto, ou o tratamento realizado pela rotina é o mais adequado para aquele volume de dados.

O exemplo clássico desta situação é para importação de lançamentos ou saldos contábeis. A aplicação ERP possui uma rotina de integração denominada “Contabilização TXT” (CTBA500) mas a rotina de lançamentos contábeis (CTBA102) pode ser executada através da MsExecAuto.

Para importações de saldos e lançamentos deve-se optar pela utilização da rotina de “Contabilização TXT”, devido aos tratamentos que a mesma permite através das configurações da própria rotina em conjunto com as parametrizações do módulo. O uso da CTBA102 – Laçamentos Contábeis via MsExecAuto é recomendado para rotinas de manutenção ou funcionalidades específicas / melhorias, com por exemplo o estorno de lançamentos contábeis sem a necessidade de avaliar regras de contabilização.

41.1. Desenvolvendo aplicações com MsExecAuto

Estrutura de uma rotina com execução de MsExecAuto

- ☑ Definição das variáveis de controle da MsExecAuto
- ☑ Montagem dos arrays de parâmetros
- ☑ Definição dos parâmetros específicos da rotina que será executada
- ☑ Controle de transação
- ☑ Tratamento do sucesso ou não da operação

Variáveis de controle

☑ **Private lMsErroAuto**

Indica de se houve erro não fatal durante a execução.

☑ **Private lMsHelpAuto**

Habilita a captura das mensagens de erro.

☑ **Private lAutoErrNoFile**

Desabilita a geração do arquivo de log padrão do sistema.

Montagem dos arrays de parâmetros

☑ **Tipos de arrays de parâmetros:**

Basicamente uma MsExecAuto possui dois tipos de arrays de parâmetros: Array de Dados de Cabeçalho e Array de Dados de Itens.

Os exemplos a seguir visam ilustrar a utilização de cada uma dos tipos de arrays:

- ❒ Cadastro de clientes (MATA030): Esta rotina atualiza apenas os dados da tabela SA1 – Cadastro de Clientes, portanto será necessário apenas um array de dados contendo as informações que deverão ser gravadas na tabela SA1.

- ❒ Documentos de entrada (MATA103): Esta rotina atualiza basicamente os dados das tabelas SF1 – Cabeçalho do Documento de Entrada e SD1 – Itens do Documento de entrada, portanto serão necessários dois arrays de dados contendo as informações do cabeçalho do documento de entrada e dos itens do documento de entrada, sendo que este último terá sua estrutura de conteúdo repetida para cada item do documento.

- ❒ Aviso de Recebimento de Carga (MATA145): Esta rotina atualiza basicamente os dados das tabelas DB1 – Aviso de Recebimento de Carga, DB2 – Cabeçalho do documento recebido e DB3 – Itens do documento recebido, portanto são necessários três array de dados contendo as respectivas informações para as tabelas DB1, DB2 e DB3, sendo que as duas últimas terão suas estruturas de conteúdo repetidas para cada documento recebido.

☑ **Estrutura dos arrays de parâmetros:**

❒ **Array de dados do cabeçalho (xAutoCab):**

O array contendo os dados do cabeçalho é composto por: Campo, Conteúdo e Validação.

Cada item deste array é um dos campos da tabela de cabeçalho das informações que serão processadas pela MsExecAuto. Com base nos exemplos anteriores, a MsExecAuto da MATA030 possui apenas um array de parâmetros no formato xAutoCab, sendo que a MATA103 utilizará o xAutoCab para a tabela SF1 e a MATA145 para a tabela DB1.

Sintaxe:

```
AADD(xAutoCab,{“Campo”, xConteúdo, “Validação”})
```

Exemplo:

```
AADD(xAutoCab,{“A1_FILIAL”    , xFilial(“SA1”)  , Nil})
AADD(xAutoCab,{“A1_COD”       , “000001”        , Nil})
AADD(xAutoCab,{“A1_LOJA”      , “01”            , Nil})
AADD(xAutoCab,{“A1_NOME”      , “TESTE-000001”  , Nil})

MsExecAuto({|x,y| MATA030(x,y)}, xAutoCab, 3)
```

❒ **Array de dados dos itens (xAutoItens):**

O array contendo os dados dos itens tambémé composto por: Campo, Conteúdo e Validação; sendo que a diferença está no fato de que para cada item a ser tratado pela MsExecAuto deverá ser repetida e preenchida a estrutura do array com as informações do mesmo.

Com base nos exemplos anteriores, a MsExecAuto da MATA030 não possui um array de parâmetros no formato xAutoItens, já a MATA103 utilizará o xAutoItens para a tabela SD1 e a MATA145 utilizará dois xAutoItens, sendo o primeiro para a tabela DB2 e o segundo para a tabela DB3.

Sintaxe:

```
<Estrutura de controle e repetição>

      xAutoItem := {}

      AADD(xAutoItem,{“Campo”, xConteúdo, “Validação”})

      AADD(xAutoItens, xAutoItem)

</Estrutura>
```

Exemplo:

```
AADD(xAutoCab,{“F1_FILIAL”    , xFilial(“SF1”)  , Nil})
AADD(xAutoCab,{“F1_DOC”       , “000001”        , Nil})
AADD(xAutoCab,{“F1_SERIE”     , “1”             , Nil})
AADD(xAutoCab,{“F1_FORNECE”   , “000001”        , Nil})
AADD(xAutoCab,{“F1_LOJA”      , “01”            , Nil})

For nX := 1 to 2

      xAutoItem := {}

      AADD(xAutoItem,{“D1_FILIAL”   , xFilial(“SD1”)  , Nil})
      AADD(xAutoItem,{“D1_DOC”      , “000001”        , Nil})
      AADD(xAutoItem,{“D1_SERIE”    , “1”             , Nil})
      AADD(xAutoItem,{“D1_FORNECE”  , “000001”        , Nil})
      AADD(xAutoItem,{“D1_LOJA”     , “01”            , Nil})
      AADD(xAutoItem,{“D1_ITEM”     , STRZERO(nx,04)  , Nil})
      AADD(xAutoItem,{“D1_COD”      , STRZERO(nx,15)  , Nil})
      AADD(xAutoItem,{“D1_QUANT”    , 100*nX          , Nil})
      AADD(xAutoItem,{“D1_VUNIT”    , 5*nX            , Nil})
      AADD(xAutoItem,{“D1_TOTAL”    , (100*nX)*(5*nX) , Nil})
      AADD(xAutoItem,{“D1_TES”      , “001”           , Nil})

      AADD(xAutoItens, xAutoItem)

Next nX

MsExecAuto({|x,y,z| MATA103(x,y,z)}, xAutoCab, xAutoItens, 3)
```

☑ **Método de atribuição de conteúdo Direto:**

O método direto consiste em atribuir diretamente na montagem do array de parâmetros o conteúdo de cada campo que será tratado pela MsExecAuto.

Este método é o mais comum de ser utilizado dado sua praticidade, e pela possibilidade do desenvolvedor especificar somente os campos obrigatórios para a tabela e aqueles que conterão as informações geradas por sua aplicação.

Exemplo:

```
AADD(xAutoCab,{“A1_FILIAL”    , xFilial(“SA1”)  , Nil})
AADD(xAutoCab,{“A1_COD”       , “000001”        , Nil})
AADD(xAutoCab,{“A1_LOJA”      , “01”            , Nil})
AADD(xAutoCab,{“A1_NOME”      , “TESTE-000001”  , Nil})

MsExecAuto({|x,y| MATA030(x,y)}, xAutoCab, 3)
```

☑ **Método de atribuição de conteúdo pela estrutura:**

O método de atribuição pela estrutura é recomendável em rotinas que efetuam a importação das informações através da leitura de arquivos de dados, sejam eles CODEBASE ou via banco de dados.

Este método consiste em comparar a estrutura do arquivo de origem (a ser importado) contra a do arquivo de destino (tratado pela MsExecAuto). Desta forma os campos que forem coincidentes entre o arquivo de origem e o arquivo de destino serão preenchidos com as informações do arquivo importado, caso contrário os mesmos poderão ser desconsiderados ou serem preenchidos com um conteúdo padrão.

Exemplo:

```
// Captura a estrutura da tabela padrão do sistema
DbSelectArea("SN1")
DbSetOrder(1)
aStruSN1 := SN1->(DbStruct())

// Efetua a abertura do arquivo SN1 que será importado
cArqSN1:= cDirArq+"SN1IMP"
IF File(cArqSN1+cFileExt)
      dbUseArea(.T.,,cArqSN1+cFileExt,"SN1IMP",.F.,.F.)
      IndRegua("SN1IMP",cArqSN1+OrdBagExt(),"N1_FILIAL+N1_CBASE+N1_ITEM",;
      ,,"Selecionando Registros...")
ELSE
      HELP("IATF001",1,"HELP","NO_FILE","ARQUIVO SN1IMP não existe."+;
      CRLF+"Verifique caminho informado.",1,0)
      RETURN
ENDIF

// Efetua a leitura do arquivo a ser importado
DbSelectArea("SN1IMP")
DbGotop()
WHILE SN1IMP->(!Eof())

      // Efetua a montagem do xAutoCab com base nas estruturas dos
      // arquivos

      FOR nX := 1 TO LEN(aStruSN1)
            IF SN1IMP->(FieldPos(aStruSN1[nX][1]))>0
                  DO CASE
                        CASE EMPTY(SN1IMP->&(aStruSN1[nX][1]))
                              AADD(xAutoCab,{aStruSN1[nX][1],;
                                    CRIAVAR(aStruSN1[nX][1]), NIL})

                        CASE aStruSN1[nX][2] == "C"
                              AADD(xAutoCab,{aStruSN1[nX][1],;
                                    SN1IMP->&(aStruSN1[nX][1]),;
                                    "",aStruSN1[nX][3]), NIL})

                        CASE aStruSN1[nX][2] == "N"
                              AADD(xAutoCab,{aStruSN1[nX][1],;
                                    ABS(SN1IMP->&(aStruSN1[nX][1])), NIL})
```

```
                        OTHERWISE
                              AADD(xAutoCab,{aStruSN1[nX][1],;
                                   SN1IMP->&(aStruSN1[nX][1]), NIL})
                  ENDCASE
            ELSE
                  AADD(xAutoCab,{aStruSN1[nX][1],;
                       CRIAVAR(aStruSN1[nX][1]), NIL})
            ENDIF
      NEXT nX

      ...

      MsExecAuto({|x,y,z| ATFA010(x,y,z)}, xAutoCab, xAutoItens, 3)

END
```

Definição dos parâmetros específicos da rotina que será executada

Este é ponto no qual o desenvolvedor inclui os tratamentos necessários a correta execução da rotina, além de seus tratamentos específicos, os quais justificaram o desenvolvimento.

Utilizando o mesmo exemplo anterior da ATFA010 – Cadastro de Ativos podemos citar dois tratamentos específicos necessários para a correta execução da MsExecAuto:

- **Chamada das perguntas da ATFA010:**

  A execução da função ***Pergunte("ATFA010",.F.)*** permite que seja definidos os conteúdos desejados para os parâmetros da rotina de cadastro conforme a necessidade da aplicação, como por exemplo permitir a inclusão de uma chapa de bem em branco ou não efetuar a contabilização da inclusão.

- **Atualização das variavéis de memória do ALIAS "M->" para o SN1:**

  A execução da função ***RegToMemory("SN1",.T.)*** é necessária para a utilização da função CRIAVAR() no método de atribuição pela estrutura.

Muitas vezes uma estrutura que funciona para um MsExecAuto pode não funcionar para outro do mesmo tipo, devido as características individuais de cada rotina. Podemos citar os seguintes exemplos:

- **MATA103** (Documento de Entrada): Para tratamento do título financeiro a ser gerado pela nota fiscal quando o parâmetro de atualização de financeiro da TES está ativado, deve ser incluído o seguinte item no array xAutoCab:

  {" E2_NATUREZ " ,"NAT01" ,NIL})

- **MATA650** (Abertura de ordem de produção): Para que sejam geradas as ordens de produção intermediárias na abertura de uma ordem de produção principal deve ser incluído o seguinte item no array xAutoCab:

  {"AUTEXPLODE" ,"S" ,NIL})

Controle de transação

A utilização do controle de transação permite garantir a integridade das informações gravadas pelo processamento.

Para utilização do controle de transação na aplicação podem ser utilizados os seguintes blocos de controle:

☑ **Begin Transaction ... DisarmTransaction() ... End Transaction**

Um bloco de código determinado pelos comandos Begin Transaction ... End Transaction terá suas informações atualizadas somente se antes do execução do comando End Transaction não for executada a função DisarmTransaction().

Desta forma pode-se implementar um controle de transação por item processado, sendo executado o DisarmTransaction() para aquele elemento em que houve algum problema de processamento.

Seguindo este mesmo raciocínio, caso ocorra um erro fatal na aplicação, somente o item que está em processamento será desconsiderado, pois a aplicação ERP efetua um DisarTransaction() automaticamente, fechamento as transações pendentes e restaurando as situações anteriores, mas apenas para aqueles processamento protegidos pelo bloco de controle de transação.

O comando END TRANSACTION não pode ter sua interpretação vinculada a uma condição. Nestes casos ocorrerá um erro de compilação indicando que o bloco aberto pelo comando BEGIN TRANSACTION não foi corretamente fechado.

Exemplo:

```
AADD(xAutoCab,{“A1_FILIAL”    , xFilial(“SA1”)  , Nil})
AADD(xAutoCab,{“A1_COD”       , “000001”        , Nil})
AADD(xAutoCab,{“A1_LOJA”      , “01”            , Nil})
AADD(xAutoCab,{“A1_NOME”      , “TESTE-000001”  , Nil})

BEGIN TRANSACTION

lMsErroAuto := .F.
MsExecAuto({|x,y| MATA030(x,y)}, xAutoCab, 3)

IF lMsErroAuto
      DisarmTransaction()
ENDIF

END TRANSACTION
```

☑ **BeginTran() ... DisarmTransaction() ... EndTran()**

As funções BeginTran() e EndTran() visam permitir que o término da transação seja condicional, eliminando a limitação da estrutura BEGIN TRANSACTION ... END TRANSACTION.

Exemplo:

```
AADD(xAutoCab,{"A1_FILIAL"    , xFilial("SA1")  , Nil})
AADD(xAutoCab,{"A1_COD"       , "000001"        , Nil})
AADD(xAutoCab,{"A1_LOJA"      , "01"            , Nil})
AADD(xAutoCab,{"A1_NOME"      , "TESTE-000001"  , Nil})

BeginTran()

lMsErroAuto := .F.
MsExecAuto({|x,y| MATA030(x,y)}, xAutoCab, 3)

IF lMsErroAuto
      DisarmTransaction()
ELSE
      EndTran()
ENDIF

MsUnlockAll()
```

Importante

Neste modelo de controle de transação é recomendável a utilização da função MsUnlockAll() para destravar todos os registros que estejam eventualmente travados.

Anotações

Tratamento de mensagens de erro

Com a utilização da funcionalidade MsExecAuto a aplicação ERP disponibiliza diversos métodos para visualização e tratamento das mensagens de erro ocorridas durante a execução da rotina, sendo estes:

- ❒ Visualização do evento de erro;
- ❒ Gravação do evento de erro em arquivo texto;
- ❒ Personalização da gravação do evento de erro.

☑ Visualização do evento de erro

Para visualização em tela do evento de erro ocorrido durante o processamento da rotina via MsExecAuto deve-se utilizar a função MostraErro(), conforme o exemplo:

```
Private lMsHelpAuto     := .T.
Private lAutoErrNoFile  := .F.

AADD(xAutoCab,{"A1_FILIAL"    , xFilial("SA1")  , Nil})
AADD(xAutoCab,{"A1_COD"       , "000001"        , Nil})
AADD(xAutoCab,{"A1_LOJA"      , "01"            , Nil})
AADD(xAutoCab,{"A1_NOME"      , "TESTE-000001"  , Nil})

BEGIN TRANSACTION

lMsErroAuto := .F.
MsExecAuto({|x,y| MATA030(x,y)}, xAutoCab, 3)

IF lMsErroAuto
      MostraErro()
      DisarmTransaction()
ENDIF

END TRANSACTION
```

O conteúdo das variáveis PRIVATE de controle da MsExecAuto deve ser configurado conforme abaixo:

```
Private lMsHelpAuto     := .T.
Private lAutoErrNoFile  := .F.
```

☑ **Gravação do evento de erro em arquivo texto**

Para gravação em arquivo no formato texto do evento de erro ocorrido durante o processamento da rotina via MsExecAuto deve-se utilizar a função MostraErro(), conforme o exemplo:

```
Private lMsHelpAuto     := .T.
Private lAutoErrNoFile  := .F.

AADD(xAutoCab,{"A1_FILIAL"    , xFilial("SA1")  , Nil})
AADD(xAutoCab,{"A1_COD"       , "000001"        , Nil})
AADD(xAutoCab,{"A1_LOJA"      , "01"            , Nil})
AADD(xAutoCab,{"A1_NOME"      , "TESTE-000001"  , Nil})

BEGIN TRANSACTION

lMsErroAuto := .F.
MsExecAuto({|x,y| MATA030(x,y)}, xAutoCab, 3)

IF lMsErroAuto
      MostraErro("\system\")
      DisarmTransaction()
ENDIF

END TRANSACTION
```

Dica

A função MostraErro() possui o parâmetro cPath o qual pode ser informado de duas formas:

- Apenas o diretório: Se for informado apenas o diretório será gerado um arquivo com denominação no formato SCxxxxxx.log, aonde xxxxxx será um número sequencial gerado internamente pelo sistema.
- Informando o diretório e o nome do arquivo: A função irá respeitar o nome do arquivo informado, sobrescrendo o conteúdo anterior.

Dica

Caso seja necessário gerar um único arquivo texto contendo todos os eventos de erro gerados pelos sucessivos processamentos da MsExecAuto, a função MostraErro(cPath) deverá ser chamada apenas uma vez, ao término do processamento da rotina.

Isto ocorre pois a função MostraErro() limpa o cache de eventos de erros controlado pela MsExecAuto.

☑ **Personalização da gravação do evento de erro**

Para processamentos mais volumosos, a geração de diversos arquivos de textos ou até mesmo a geração de um único arquivo texto contendo todos os eventos de erro pode dificultar a análise e correção dos problemas encontrados durante o processamento.

Desta forma é possível personalizar a gravação do evento de erro, de forma que o mesmo seja gerado em um arquivo no formato .DBF, permitindo o vínculo do registro processado com a mensagem de erro gerada.

```
Private lMSHelpAuto     := .F.
Private lAutoErrNoFile  := .T.

AADD(xAutoCab,{"A1_COD"       , "000001"        , Nil})
AADD(xAutoCab,{"A1_LOJA"      , "01"            , Nil})
AADD(xAutoCab,{"A1_NOME"      , "TESTE-000001"  , Nil})

// Função específica que cria o arquivo no formato DBF que conterá as
// mensagens de erro.
XDBFLOG()

BEGIN TRANSACTION

lMsErroAuto := .F.
MsExecAuto({|x,y| MATA030(x,y)}, xAutoCab, 3)

IF lMsErroAuto
      // Função que retorna o evento de erro na forma de um array
      aAutoErro := GETAUTOGRLOG()

      // Função especifica que converte o array aAutoErro em texto
      // contínuo, com a quantidade de caracteres desejada por linha

      // Função específica que efetua a gravação do evento de erro no
      // arquivo previamente crado.
      XGRVLOG(XCONVERRLOG(aAutoErro))

      DisarmTransaction()
ENDIF

END TRANSACTION
```

As funções específicas XDBFLOG(), XCONVERRLOG() e XGRVLOG() serão detalhadas no exemplo completo de utilização da função MsExecAuto().

Aplicação completa de importação utilizando MsExecAuto: Carga de imobilizado

```
#INCLUDE "PROTHEUS.CH"

/*/
+-----------------------------------------------------------------------
| Função    | IATF001       | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | IMPORTACAO DO CADASTRO DE ATIVO IMOBILIZADO              |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                              |
+-----------------------------------------------------------------------
/*/

USER FUNCTION IATF001()

LOCAL oProcess

PRIVATE oMainWnd

//Perguntas para parametrizacao da rotina (PARAMBOX)
//MVParBox01 - Tipo de informação (.DBF/.TXT)
//MVParBox02 - LayOut do arquivo  (Padrão / Especifico)
//MVParBox03 - Arquivo de dados   (Consulta)
//MVParBox04 - Arquivo de layout  (Consulta)
//MVParBox05 - Operacao           (Incluir, Alterar, Excluir)
//MVParBox06 - Diretorio de Log   (Consulta)
//MVParBox07 - Importa CIAP       (SIM/NAO)

PRIVATE aTiposFile := {".DBF",".DTC",".TXT"}
PRIVATE MvParBox01 := 0
PRIVATE MvParBox02 := 0
PRIVATE MvParBox03 := ""
PRIVATE MvParBox04 := ""
PRIVATE MvParBox05 := 0
PRIVATE MvParBox06 := ""
PRIVATE MVParBox07 := 0

PRIVATE _cDirectory  := ""

RpcSetType(3)
RpcSetEnv("99", "01",,,,,
{"CT1","SF9","SN1","SN2","SN3","SN4","SN5","SNG","SM2"})

SET DATE FORMAT "dd/mm/yyyy"

oMainWnd := TWindow():New( 000, 000, 001, 001, "Importação: Fichas do
imobilizado",,,,,,,,,,,,,,,,.T.)
oMainWnd:bInit := {|| IIF(IC001PARBX(),(oProcess:= MsNewProcess():New({|lEnd|
PROCATIVO(.F.,oProcess)}),oProcess:Activate(),oMainWnd:End()),oMainWnd:End())}
oMainWnd:Activate("ICONIZED")

RpcClearEnv()
RETURN
```

Continuação:

```
/*/
+-----------------------------------------------------------------------
| Função    | PROCATIVO     | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | PROCESSAMENTO DA IMPORTACAO DO CADASTRO DE IMOBILIZADO   |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

STATIC FUNCTION PROCATIVO(lEnd,oProcess)

Local aArea             := GetArea()
Local aDadosSN1         := {}
Local aDadosSN3         := {}
Local aDadosSF9         := {}
Local aDadosTN1         := {}
Local aDadosTN2         := {}
Local aDadosTN3         := {}
Local aItemSN3          := {}
Local cArqSN1           := ""
Local cArqSN2           := ""
Local cArqSN3           := ""
Local cArqSA2           := ""
Local aStruSN1          := {}
Local aStruSN2          := {}
Local aStruSN3          := {}
Local aStruSF9          := {}
Local nX                := 0
Local nTotRegs          := 0
Local nProcRegs         := 0
Local nPosDados         := 0
Local cCodFor           := ""
Local cLojaFor          := ""
Local cDescFor          := ""
Local cFilAtu           := cFilAnt
Local cDirArq           := Alltrim(MVParBox03)
Local cHistorico        := ""
Local nX                := 0
Local cFileExt          := aTiposFile[MvParBox01]
Local lImpSF9           := MvParBox07==1
Local lImpSNG           := .F. // Ainda nao implementado
Local cKeyImp           := ""
Local aAutoErro         := {}

// Tratamentos adicionais - arquivos especificos do cliente
Local cArqCTA           := ""
Local cArqSM2           := ""
Local cIndSNG
Local nIndex            := 0
Local cGrupo            := ""
Local cContab           := ""
Local aEntSNG           := {}
Local cChapa            := "000000"

Private lMsErroAuto     := .F.
Private lMSHelpAuto     := .F.
```

Continuação:

```
Private lAutoErrNoFile  := .T.

// Garante a abertura dos arquivos padrões do sistema
DbSelectArea("CT1")
DbSetOrder(2) // FILIAL + COD.REDUZIDO

DbSelectArea("SM2")
DbSetOrder(1)

DbSelectArea("SN1")
DbSetOrder(1)
aStruSN1 := SN1->(DbStruct())

DbSelectArea("SN2")
DbSetOrder(1)
aStruSN2 := SN2->(DbStruct())

DbSelectArea("SN3")
DbSetOrder(1)
aStruSN3 := SN3->(DbStruct())

DbSelectArea("SF9")
DbSetOrder(1)
aStruSF9 := SF9->(DbStruct())

// Prepara o arquivo no formato .DBF para gravação dos eventos de erro
XDBFLOG()

DbSelectArea("SNG")
cIndSNG := CRIATRAB(Nil,.F.)
IndRegua("SNG",cIndSNG,"NG_FILIAL+NG_CCONTAB",,,"Selecionando Registros...")
nIndex := RetIndex()
#IFNDEF TOP
      DbSetIndex(cIndSNG+OrdBagExt())
#ENDIF
DbSetOrder(nIndex)

PERGUNTE("ATF010",.F.)
MV_PAR02 := 1 // Permite repetir chapa definido como sim

IF Select("SN1IMP") > 0
      DbSelectArea("SN1IMP")
      DbCloseArea()
ENDIF
IF Select("SN2IMP") > 0
      DbSelectArea("SN2IMP")
      DbCloseArea()
ENDIF
IF Select("SN3IMP") > 0
      DbSelectArea("SN3IMP")
      DbCloseArea()
ENDIF
IF Select("SF9IMP") > 0
      DbSelectArea("SF9IMP")
      DbCloseArea()
ENDIF
```

Continuação:

```
IF Select("CTAIMP") > 0
      DbSelectArea("CTAIMP")
      DbCloseArea()
ENDIF

cArqSN1:= cDirArq+"SN1IMP"
IF File(cArqSN1+cFileExt)
      dbUseArea(.T.,,cArqSN1+cFileExt,"SN1IMP",.F.,.F.)
      IndRegua("SN1IMP",cArqSN1+OrdBagExt(),"N1_FILIAL+N1_CBASE+N1_ITEM",,,;
               "Selecionando Registros...")
ELSE
      HELP("IATF001",1,"HELP","NO_FILE","ARQUIVO SN1IMP não existe."+CRLF+;
                 "Verifique caminho informado.",1,0)
      RETURN
ENDIF

cArqSN2:= cDirArq+"SN2IMP"
IF File(cArqSN2+cFileExt )
      dbUseArea(.T.,,cArqSN2+cFileExt,"SN2IMP",.F.,.F.)
      IndRegua("SN2IMP",cArqSN2+OrdBagExt(),;
      "N2_FILIAL+N2_CBASE+N2_ITEM+N2_TIPO+N2_SEQ",,,"Selecionando Registros...")
ELSE
      HELP("IATF001",1,"HELP","NO_FILE","ARQUIVO SN2IMP não existe."+CRLF+;
                 "Verifique caminho informado.",1,0)
      RETURN
ENDIF

cArqSN3:= cDirArq+"SN3IMP"
IF File(cArqSN3+cFileExt)
      dbUseArea(.T.,,cArqSN3+cFileExt,"SN3IMP",.F.,.F.)
      IndRegua("SN3IMP",cArqSN3+OrdBagExt(),"N3_FILIAL+N3_CBASE+N3_ITEM",,,;
               "Selecionando Registros...")
ELSE
      HELP("IATF001",1,"HELP","NO_FILE","ARQUIVO SN3IMP não existe."+CRLF+;
                 "Verifique caminho informado.",1,0)
      RETURN
ENDIF

IF lImpSF9
      cArqSF9:= cDirArq+"SF9IMP"
      IF File(cArqSF9+cFileExt )
            dbUseArea(.T.,,cArqSF9+cFileExt,"SF9IMP",.F.,.F.)
            IndRegua("SF9IMP",cArqSF9+OrdBagExt(),"F9_FILIAL+F9_CODIGO",,,;
                     "Selecionando Registros...")
      ELSE
            HELP("IATF001",1,"HELP","NO_FILE","ARQUIVO SF9IMP não existe."+;
                 CRLF+"Verifique caminho informado.",1,0)
            RETURN
      ENDIF
ENDIF
```

Continuação:

```
cArqCTA:= cDirArq+"CTAIMP"
IF File(cArqCTA+cFileExt)
      dbUseArea(.T.,,cArqCTA+cFileExt,"CTAIMP",.F.,.F.)
      IndRegua("CTAIMP",cArqCTA+OrdBagExt(),"CODIGO_CON",,,;
               "Selecionando Registros...")
ELSE
      HELP("IATF001",1,"HELP","NO_FILE","ARQUIVO CTAIMP não existe."+CRLF+;
                 "Verifique caminho informado.",1,0)
      RETURN
ENDIF

cArqSM2:= cDirArq+"SM2IMP"
IF File(cArqSM2+cFileExt)
      dbUseArea(.T.,,cArqSM2+cFileExt,"SM2IMP",.F.,.F.)
      IndRegua("SM2IMP",cArqSM2+OrdBagExt(),"DTOS(BTN_DATA)",,,"Selecionando
Registros...")
ELSE
      HELP("IATF001",1,"HELP","NO_FILE","ARQUIVO SM2IMP não existe."+CRLF+;
                 "Verifique caminho informado.",1,0)
      RETURN
ENDIF

lMSHelpAuto := .T.
DbSelectArea("SN1IMP")
DbGotop()

//ÚÄÄÄ
```

Continuação:

```
While SN1IMP->(!Eof())

      nProcRegs++
      oProcess:IncRegua1("Processando item: "+CValToChar(nProcRegs)+" / "+;
                         CValToChar(nTotRegs))
      oProcess:IncRegua2("Ficha de Ativo: ")

      aDadosSN1 := {}
      aDadosSN3 := {}

      // Compatibilização para utilização do CRIAVAR()
      REGTOMEMORY("SN1",.T.)

      //Monstagem do array com dados sem tratamento e com as informação da IMP

      FOR nX := 1 TO LEN(aStruSN1)
            IF SN1IMP->(FieldPos(aStruSN1[nX][1]))>0
                  DO CASE
                        CASE EMPTY(SN1IMP->&(aStruSN1[nX][1]))
                              AADD(aDadosSN1,{aStruSN1[nX][1],;
                                    CRIAVAR(aStruSN1[nX][1]), NIL})
                        CASE aStruSN1[nX][2] == "C"
                              AADD(aDadosSN1,{aStruSN1[nX][1],;
                                    SN1IMP->&(aStruSN1[nX][1]),;
                                    "",aStruSN1[nX][3]), NIL})
                        CASE aStruSN1[nX][2] == "N"
                              AADD(aDadosSN1,{aStruSN1[nX][1],;
                                    ABS(SN1IMP->&(aStruSN1[nX][1])), NIL})
                        OTHERWISE
                              AADD(aDadosSN1,{aStruSN1[nX][1],;
                                    SN1IMP->&(aStruSN1[nX][1]), NIL})
                  ENDCASE
            ELSE
                  AADD(aDadosSN1,{aStruSN1[nX][1], CRIAVAR(aStruSN1[nX][1]),;
                        NIL})
            ENDIF
      NEXT nX

      // Alteração das informações do array de acordo com a necessidade
      // AADD(aDadosTN1,{"CAMPO",CONTEUDO,"VALIDACAO"})
      aDadosTN1   := {}

      // ALTERACAO DO ARRAY ADADOSSN1 COM AS INFORMACOES TRABALHADAS E
      // ARMAZENADAS EM ADADOSTN1
      // AADD(aDadosTN1,{"N1_XXXXXX"           ,xConteudo        ,Nil})

      For nX := 1 TO LEN(aDadosTN1)
            IF (nPosField := aScan(aDadosSN1,{|aDadoSN1| aDadosSN1[1] ==;
                aDadosTN1[nX][1]})) > 0
                  aDadosSN1[nPosField][2] := aDadosTN1[nX][2]
                  aDadosSN1[nPosField][3] := aDadosTN1[nX][3]
            ENDIF
      Next nX
```

Continuação:

```
      IF SN1->(DbSeek(aDadosSN1[1][2]+aDadosSN1[2][2]+aDadosSN1[3][2]))
            ConOut("Registro já importado: "+;
             aDadosSN1[1][2]+"/"+aDadosSN1[2][2]+"/"+aDadosSN1[3][2])
            ConOut("Registros importados com sucesso: "+CValToChar(nProcRegs))
            SN1IMP->(dbSkip())
            Loop
      ENDIF

      SN3IMP->(DbSeek(SN1IMP->(N1_FILIAL+N1_CBASE+N1_ITEM)))
      While SN3IMP->(!Eof()) .AND. SN3IMP->(N3_FILIAL+N3_CBASE+N3_ITEM) ==;
            SN1IMP->(N1_FILIAL+N1_CBASE+N1_ITEM)

            aItensSN3 := {}

            //³ Monstagem do array com dados sem tratamento e com as informações

            FOR nX := 1 TO LEN(aStruSN3)
                  IF SN3IMP->(FieldPos(aStruSN3[nX][1]))>0
                        DO CASE
                              CASE EMPTY(SN3IMP->&(aStruSN3[nX][1]))
                                    AADD(aItensSN3,{aStruSN3[nX][1],;
                                          CRIAVAR(aStruSN3[nX][1]), NIL})
                              CASE aStruSN3[nX][2] == "C"
                                     AADD(aItensSN3,{aStruSN3[nX][1],;
                                          SN3IMP->&(aStruSN3[nX][1]),;
                                          aStruSN3[nX][3]), NIL})
                              CASE aStruSN3[nX][2] == "N"
                                    AADD(aItensSN3,{aStruSN3[nX][1],;
                                          ABS(SN3IMP->&(aStruSN3[nX][1])),;
                                          ".T."})

                              OTHERWISE
                                    AADD(aItensSN3,{aStruSN3[nX][1],;
                                          SN3IMP->&(aStruSN3[nX][1]), NIL})

                        ENDCASE
                  ELSEIF aStruSN3[nX][2] == "N"
                        AADD(aItensSN3,{aStruSN3[nX][1],;
                              CRIAVAR(aStruSN3[nX][1]), ".T."})
                  ELSE
                        AADD(aItensSN3,{aStruSN3[nX][1],;
                              CRIAVAR(aStruSN3[nX][1]), NIL})
                  ENDIF
            NEXT nX

          // Alteração das informações do array de acordo com a necessidade
          // AADD(aDadosTN3,{"CAMPO"      ,CONTEUDO,"VALIDACAO"})
             aDadosTN3  := {}

          // ALTERACAO DO ARRAY aItenSN3  COM AS INFORMACOES TRABALHADAS E
          // ARMAZENADAS EM aDadosTN3

            For nX := 1 TO LEN(aDadosTN3)
                  IF (nPosField := aScan(aItensSN3,{|aItenSN3| aItenSN3[1] ==;
                      aDadosTN3[nX][1]})) > 0
                        aItensSN3[nPosField][2] := aDadosTN3[nX][2]
                        aItensSN3[nPosField][3] := aDadosTN3[nX][3]
```

Continuação:

```
                ENDIF
            Next nX

            AADD(aDadosSN3,aItensSN3)
            SN3IMP->(DbSkip())
      Enddo

      IF    Len(aDadosSN1) > 0 .AND. Len(aDadosSN3) > 0

            ConOut("Iniciando MsExecAuto - ATFA010: "+Time())

            Begin Transaction
            cFilAnt := SN1IMP->N1_FILIAL
            lMsErroAuto := .F.
            MSExecAuto({|x,y,z| Atfa010(x,y,z)},aDadosSN1,aDadosSN3,3)
            // Cabeçalho, Itens e Opção

            ConOut("Finalizada MsExecAuto - ATFA010: "+Time())

            IF lMsErroAuto
                  aAutoErro := GETAUTOGRLOG()
                  XGRVLOG(cKeyImp,;
                          SN1IMP->(N1_FILIAL+N1_CBASE+N1_ITEM),;
                          SN1IMP->N1_DESCRIC, XCONVERRLOG(aAutoErro))
                  DisarmTransaction()
                  MostraErro(Alltrim(MVParBox06))
            ELSE

                  // TRATAMENTO DA DESCRIÇÃO ESTENDIDA (SN2)
                  SN2IMP->(DbSeek(SN1IMP->(N1_FILIAL+N1_CBASE+N1_ITEM)))
                  cHistorico := ALLTRIM(SN2IMP->N2_HISTOR1) +;
                                ALLTRIM(SN2IMP->N2_HISTOR2)

                  For nX := 1 to Len(cHistorico) STEP 40

                        RECLOCK("SN2",.T.)
                        SN2->N2_FILIAL    := SN1->N1_FILIAL
                        SN2->N2_CBASE     := SN1->N1_CBASE
                        SN2->N2_ITEM      := SN1->N1_ITEM
                        SN2->N2_TIPO      := "01"
                        SN2->N2_SEQ       := "001"
                        SN2->N2_SEQUENC   := STRZERO(nX,2)
                        SN2->N2_HISTOR    := SUBSTR(cHistorico,nX,40)
                        MSUNLOCK()

                  NEXT nX

                  // TRATAMENTO DA CARGA DO CIAP
                  IF lImpSF9
                     IATFCIAP(aStruSF9,aDadosSF9)
                  ENDIF

                  ConOut("Registros importados com sucesso: "+;
                         CValToChar(nProcRegs))

            ENDIF
            End Transaction
```

Continuação:

```
      ENDIF

      SN1IMP->(DbSkip())
Enddo

oProcess:IncRegua1("Processamento finalizado")

DbSelectArea("SN1IMP")
DbCloseArea()
IF File(cArqSN1+OrdBagExt())
      FErase(cArqSN1+OrdBagExt())
ENDIF

DbSelectArea("SN2IMP")
DbCloseArea()
IF File(cArqSN2+OrdBagExt())
      FErase(cArqSN2+OrdBagExt())
ENDIF

DbSelectArea("SN3IMP")
DbCloseArea()
IF File(cArqSN3+OrdBagExt())
      FErase(cArqSN3+OrdBagExt())
ENDIF

IF lImpSF9
      DbSelectArea("SF9IMP")
      DbCloseArea()
      IF File(cArqSF9+OrdBagExt())
            FErase(cArqSF9+OrdBagExt())
      ENDIF
ENDIF

IF lImpSNG
      DbSelectArea("SNGIMP")
      DbCloseArea()
      IF File(cIndSNG+OrdBagExt())
            FErase(cIndSNG+OrdBagExt())
      ENDIF
ENDIF

DbSelectArea("CTAIMP")
DbCloseArea()
IF File(cArqCTA+OrdBagExt())
      FErase(cArqCTA+OrdBagExt())
ENDIF

DbSelectArea("SM2IMP")
DbCloseArea()
IF File(cArqSM2+OrdBagExt())
      FErase(cArqSM2+OrdBagExt())
ENDIF
```

Continuação:

```
DbSelectArea("LOGIMP")
DbCloseArea()

ConOut("Total de registros importados: "+CValToChar(nProcRegs))
ConOut("Término da importação: "+Time())

cFilAnt := cFilAtu
RestArea(aArea)
RETURN

/*/
+-----------------------------------------------------------------------
| Função    | CT001PARBX    | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | TELA DE PARAMETROS ESPECIFICOS DA ROTINA CUSTOMIZADA     |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                              |
+-----------------------------------------------------------------------
/*/

STATIC FUNCTION IC001PARBX()

LOCAL aParamBox   := {}
LOCAL cTitulo     := "Importacao de cadastros"
LOCAL aRet        := {}
LOCAL bOk         := {|| .T.}
LOCAL aButtons    := {}
LOCAL lCentered   := .T.
LOCAL nPosx
LOCAL nPosy
LOCAL cLoad       := ""
LOCAL lCanSave    := .T.
LOCAL lUserSave   := .T.
LOCAL nX          := 0
LOCAL lRet        := .F.

AADD(aParamBox,{2,"Tipo de informação"      ,1 ,aTiposFile ,060,;
                "AllwaysTrue()", .T.})
AADD(aParamBox,{2,"LayOut do arquivo "      ,1 ,;
               {"Padrão","Especifico"},060, "AllwaysTrue()", .T.})
AADD(aParamBox,{1,"Diretorio de dados"      ,Space(60)      ,;
                "@!","AllwaysTrue()","HSSDIR" ,".T.",120,.T.})
AADD(aParamBox,{1,"Arquivo de layout "      ,Space(60)      ,;
                "@!","AllwaysTrue()","" ,".T.",120,.F.})
AADD(aParamBox,{2,"Operacao          "      ,1 ,;
                {"Incluir","Alterar","Excluir"},060, "AllwaysTrue()", .T.})
AADD(aParamBox,{1,"Diretorio de Log  "      ,Space(60)      ,;
               "@!","AllwaysTrue()","HSSDIR" ,".T.",120,.F.})
AADD(aParamBox,{2,"Importa CIAP      "      ,1 ,;
                {"Sim","Não"} ,060, "AllwaysTrue()", .T.})

lRet := ParamBox(aParamBox, cTitulo, aRet, bOk, aButtons, lCentered, nPosx,;
nPosy, /*oMainDlg*/ , cLoad, lCanSave, lUserSave)
```

Continuação:

```
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

RETURN lRet


/*/
+-----------------------------------------------------------------------
| Função    | IATFCIAP      | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | IMPORTACAO DO LIVRO FISCAL CIAP                           |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/
STATIC FUNCTION IATFCIAP(aStruSF9,aDadosSF9)
Local aDadosCIAP := {}
Local nX           := 0
Local nPosSF9      := 0

// Monta array com dados padrões do SF9 de acordo com o SX3
FOR nX := 1 to Len(aStruSF9)
      AADD(aDadosCIAP,{aStruSF9[nX][1],CRIAVAR(aStruSF9[nX][1]),NIL})
NEXT nX

// Atualiza dados do array com as informações presentes no SN1
FOR nX := 1 to Len(aDadosSF9)
      IF (nPosSF9 := aScan(aDadosCIAP,{|aLinhaCIAP| aLinhaCIAP[1] ==
aDadosSF9[nX][1]})) > 0
            aDadosCIAP[nPosSF9][2] := aDadosSF9[nX][2]
      ENDIF
NEXT nX

ConOut("Iniciando MsExecAuto - ATFCIAP: "+Time())
lMsErroAuto := .F.
MSExecAuto({|x,y| U_ATFCIAP(x,y)},aDadosCIAP,3) // Dados e Opção
ConOut("Finalizada MsExecAuto - ATFCIAP: "+Time())

RETURN lMsErroAuto
```

Continuação:

```
/*/
+-----------------------------------------------------------------------
| Função    | XDBFLOG       | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | CRIACAO DO ARQUIVO DBF PARA TRATAMENTO DOS EVENTOS DE ERR|
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/
STATIC FUNCTION XDBFLOG()
LOCAL aCampos     := {}
LOCAL cArqLog     := MVParBox06+"LOGIMP"+GetDbExtension()

IF !File(cArqLog)
      AADD(aCampos,{"CKEYIMP" ,"C",014,0})
      AADD(aCampos,{"CKEYREG" ,"C",020,0})
      AADD(aCampos,{"CDESCR"  ,"C",040,0})
      AADD(aCampos,{"CSEQMSG" ,"C",003,0})
      AADD(aCampos,{"CMSGERR" ,"C",254,0})
      dbCreate(cArqLog,aCampos,__LocalDriver)
ENDIF

dbUseArea(.T., __LocalDriver, cArqLog, "LOGIMP", .T., .F.)
RETURN

/*/
+-----------------------------------------------------------------------
| Função    | XGRVLOG       | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | GRAVACAO DA MENSAGEM DE ERRO NO ARQUIVO DBF DE CONTROLE  |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/
STATIC FUNCTION XGRVLOG(cKeyImp, cKeyReg, cDescReg, cMsgErr)

LOCAL cSeqLog := "000"

FOR nX := 1 TO Len(cMsgErr) STEP 254

      cSeqLog := SOMA1(cSeqLog)
      RECLOCK("LOGIMP",.T.)
      LOGIMP->CKEYIMP   := cKeyImp
      LOGIMP->CKEYREG   := cKeyReg
      LOGIMP->CDESCR    := cDescReg
      LOGIMP->CSEQMSG   := cSeqLog
      LOGIMP->CMSGERR   := SUBSTR(cMsgErr,nX,254)
      MSUNLOCK()

NEXT nX

RETURN
```

Continuação:

```
/*/
+-----------------------------------------------------------------------
| Função    | XCONVERRLOG   | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | CONVERTE O ARRAY AAUTOERRO EM TEXTO CONTINUO.             |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

STATIC FUNCTION XCONVERRLOG(aAutoErro)

LOCAL cRet := ""
LOCAL nX := 1

FOR nX := 1 to Len(aAutoErro)
      cRet += aAutoErro[nX]+CHR(13)+CHR(10)
NEXT nX

RETURN cRet
```

Anotações

42. Recursos de envio de e-mail

Neste tópico serão descritas as funções da linguagem ADVPL que permitem o envio e o recebimento de mensagens através de e-mails.

Funções para manipulação de e-mails

- CALLPROC
- MAILSMTPON
- MAILPOPON
- MAILSMTPOFF
- MAILPOPOFF
- MAILRECEIVE
- MAILAUTH
- POPMSGCOUNT
- MAILSEND
- MAILGETERR

Detalhamento das funções de manipulação de e-mails

CALLPROC()

Método do objeto oRpcSrv que permite a utilização de todas as funções de envio e recebimento de e-mail descritas neste tópico.

☑ **Sintaxe: CallProc(cFuncao, xParam1, ..., xParamN)**

☑ **Parâmetros:**

cFuncao	Nome da função ou método do objeto oRpcSrv que será executado.
xParamN	Parâmetros da função ou método que será executado.

☑ **Retorno:**

Nenhum	.

MAILSMTPON()

Ativa uma conexão com o serviço de SMTP para a thread ativa.

☑ **Sintaxe: MailSmtpOn(cServer, cUser, cPass, nTimeOut)**

☑ **Sintaxe clássica:**

```
CONNECT SMTP SERVER cServer ACCOUNT cUser PASSWORD cPass TIMEOUT nTimeOut IN
SERVER oRpcSrv  RESULT lResult
```

☑ **Parâmetros:**

Cserver	Endereço do servidor para conexão
Cuser	Nome do usuário para autenticação no servidor.
CPass	Senha do usuário para autenticação no servidor.
NTimeOut	Tempo de espera para operação de autenticação.

☑ **Retorno:**

Lógico	Indica se foi realizada com sucesso a conexão com o servidor indicado.

MAILPOPON()

Ativa uma conexão com o serviço de POP para a thread ativa. -> lReturn

☑ **Sintaxe: MailPopOn(cServer, cUser, cPass, nTimeOut)**

☑ **Sintaxe clássica:**

```
CONNECT POP SERVER cServer ACCOUNT cUser PASSWORD cPass  TIMEOUT nTimeOut   IN
SERVER oRpcSrv   RESULT lResult
```

☑ **Parâmetros:**

Cserver	Endereço do servidor para conexão
Cuser	Nome do usuário para autenticação no servidor.
CPass	Senha do usuário para autenticação no servidor.
NTimeOut	Tempo de espera para operação de autenticação.

☑ **Retorno:**

Lógico	Indica se foi realizada com sucesso a conexão com o servidor indicado.

MAILSMTPOFF()

Encerra a conexão com o serviço de SMTP.

☑ **Sintaxe: MailSmtpOff()**

☑ **Sintaxe clássica:**

```
DISCONNECT SMTP SERVER  IN SERVER oRpcSrv   RESULT lResult
```

☑ **Parâmetros:**

Nenhum	.

☑ **Retorno:**

Lógico	Indica se a operação foi realizada com sucesso.

MAILPOPOFF()

Encerra a conexão com o serviço de POP

☑ **Sintaxe: MailPopOff()**

☑ **Sintaxe clássica:**

```
DISCONNECT POP SERVER  IN SERVER oRpcSrv   RESULT lResult
```

☑ **Parâmetros:**

Nenhum	.

☑ **Retorno:**

Lógico	Indica se a operação foi realizada com sucesso.

MAILRECEIVE()

Efetua o recebimento de um e-mail, salvando-o no local definido.

☑ **Sintaxe: MailReceive(nNumber, @cFrom, @cTo, @cCc, @cBcc, @cSubject, @cBody, @aFiles, cPath, lDelete)**

☑ **Sintaxe clássica:**

```
RECEIVE MAIL MESSAGE nNumber FROM cFrom TO     cTo CC cCc BCC cBcc SUBJECT
cSubject BODY cBody ATTACHMENT aFiles SAVE IN cPath DELETE IN SERVER oRpcSrv
RESULT lResult
```

☑ **Parâmetros:**

nNumber	Número da mensagem que deverá ser recebida. Este número é em função da quantidade de mensagens na caixa de e-mails.
cFrom	Variável local do fonte que será atualizada com o remetente da mensagem.
cTo	Variável local do fonte que será atualizada com o destinatório da mensagem.
cCc	Variável local do fonte que será atualizada com a conta copiada na mensagem.
cBcc	Variável local do fonte que será atualizada com a conta copiada em cópia oculta na mensagem.
cSubject	Variável local do fonte que será atualizada com o assunto da mensagem.
cBody	Variável local do fonte que será atualizada com corpo da mensagem.
aFiles	Variável local do fonte que será atualizada os nomes dos anexos da mensagem.
cPath	Diretório no qual serão salvos os anexos da mensagem.
lDelete	Indica se a mensagem deverá ser apagada do servidor de e-mails após sua recepção pela aplicação.

☑ **Retorno:**

Lógico	Indica se a operação de recebimento de mensagens foi realizada com sucesso.

Anotações

MAILAUTH()

Função que executa a autenticação do usuário no serviço ativo.

☑ **Sintaxe: MailAuth(cUser, cPassword)**

☑ **Parâmetros:**

cUser	Nome do usuário para validação da conexão com o servidor.
cPassword	Senha do usuário para validação da conexão com o servidor.

☑ **Retorno:**

Lógico	Indica se foi realizada a autenticação do usuário com sucesso.

Importante

A função MailAuth() deverá ser utilizada obrigatoriamente após a abertura da conexão com o servidor, seja ele de envio ou recebimento de mensagens.

Dica

Para validação da conexão sempre efetue a verificação com o usuário contendo o endereço completo do e-mail, e caso necessário somente com o usuário, eliminando o restante do endereço após o caractere "@".

```
lResult := MailAuth(Alltrim(cEmail), Alltrim(cPass))
// Se nao conseguiu fazer a Autenticacao usando o E-mail completo, tenta
// fazer a autenticacao usando apenas o nome de usuario do E-mail.
If !lResult
        nA := At("@",cEmail)
        cUser  := If(nA>0,Subs(cEmail,1,nA-1),cEmail)
        lResult := MailAuth(Alltrim(cUser), Alltrim(cPass))
Endif
```

Anotações

POPMSGCOUNT()

Verifica quantas mensagens existem na caixa de entrada do serviço POP ativo.

☑ **Sintaxe: PopMsgCount(@nMsgCount)**

☑ **Sintaxe clássica:**

```
POP MESSAGE COUNT nMsgCount  IN SERVER oRpcSrv RESULT lResult
```

☑ **Parâmetros:**

nMsgCount	Variável local do fonte que será atualizada com a quantidade de mensagens disponíveis para recebimento na caixa de e-mails.

☑ **Retorno:**

Lógico	Indica se foi realizado o acesso a caixa de mensagens.

MAILSEND()

Envia um e-mail utilizando a serviço de SMTP ativo.

☑ **Sintaxe: MailSend(cFrom, aTo, aCc, aBcc, cSubject, cBody, aFiles, lText)**

☑ **Sintaxe clássica:**

```
SEND MAIL FROM cFrom TO aTo,... CC aCc,... BCC aBcc,... SUBJECT cSubject BODY
cBody FORMAT TEXT ATTACHMENT aFiles,... IN SERVER oRpcSrv RESULT lResult
```

☑ **Parâmetros:**

cFrom	Endereço de e-mail do remente da mensagem.
aTo	Array contendo os endereços de e-mails dos destinatários da mensagem.
aCc	Array contendo os endereços de e-mails dos copiados na mensagem.
aBcc	Array contendo os endereços de e-mails dos copiados de forma oculta na mensagem.
cSubject	Texto de assunto do e-mail.
cBody	Texto da mensagem do e-mail.
aFiles	Array contendo os nomes dos arquivos que serão anexados a mensagem.
lText	Indica se o corpo do e-mail está em formato texto.

☑ **Retorno:**

Lógico	Indica se a operação de envio de mensagens foi realizada com sucesso.

MAILGETERR()

Retorna o erro que ocorreu no envio do e-mail.

☑ **Sintaxe: MailGetErr()**

☑ **Sintaxe clássica:**

```
GET MAIL ERROR cErrorMsg IN SERVER oRpcSrv
```

☑ **Parâmetros:**

Nenhum	.

☑ **Retorno:**

Caracter	Mensagem de erro ocorrida no processamento do envio ou recebimento dos e-mails.

Exemplos de utilização das funcionalidades de envio e recebimento de e-mail

Envio de mensagens utilizando sintaxe clássica

```
#include "protheus.ch"
#include "tbiconn.ch"
#include "AP5MAIL.CH"

/*/
+-----------------------------------------------------------------------
| Função    | SENDMAIL      | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | ENVIO DE E-MAIL GENERICO                                  |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

USER FUNCTION SendMail(_lJob)

Local lResulConn := .T.
Local lResulSend := .T.
Local cError := ""

Local cServer  := AllTrim(GetMV("MV_RELSERV"))
Local cEmail   := AllTrim(GetMV("MV_RELACNT"))
Local cPass    := AllTrim(GetMV("MV_RELPSW"))
Local lRelauth := GetMv("MV_RELAUTH")

Local cDe      := cEmail
Local cPara    := "arnaldojr@microsiga.com.br"
```

Continuação:

```
Local cCc      := ""
Local cAssunto := "Teste de envio de e-mail: Curso ADVPL"
Local cAnexo   := "\SYSTEM\lgrl99.bmp"
Local cMsg     := Space(200)

Default _lJob     := .T.

cMsg  := "--------------------------------------------------"
cMsg  += "CURSO DE ADVPL                                    "
cMsg  += "--------------------------------------------------"
cMsg  += "Você está recebendo um e-mail do curso de ADVPL avançado"

CONNECT SMTP SERVER cServer ACCOUNT cEmail PASSWORD cPass RESULT lResulConn

If !lResulConn
   GET MAIL ERROR cError
   If _lJob
          ConOut(Padc("Falha na conexao "+cError,80))
   Else
      MsgAlert("Falha na conexao "+cError)
   Endif
   Return(.F.)
Endif

// Sintaxe: SEND MAIL FROM cDe TO cPara CC cCc SUBJECT cAssunto BODY cMsg
ATTACHMENT cAnexo RESULT lResulSend
// Todos os e-mail terão: De, Para, Assunto e Mensagem, porém precisa analisar
se tem: Com Cópia e/ou Anexo

If lRelauth
       lResult := MailAuth(Alltrim(cEmail), Alltrim(cPass))
   //Se nao conseguiu fazer a Autenticacao usando o E-mail completo, tenta fazer
a autenticacao usando apenas o nome de usuario do E-mail
      If !lResult
            nA := At("@",cEmail)
            cUser := If(nA>0,Subs(cEmail,1,nA-1),cEmail)
            lResult := MailAuth(Alltrim(cUser), Alltrim(cPass))
      Endif

Endif

If lResult
      If Empty(cCc) .And. Empty(cAnexo)
         SEND MAIL FROM cDe TO cPara SUBJECT cAssunto BODY cMsg RESULT
lResulSend
      Else
         If Empty(cCc) .And. !Empty(cAnexo)
                  SEND MAIL FROM cDe TO cPara SUBJECT cAssunto BODY cMsg
                  ATTACHMENT cAnexo RESULT lResulSend
         ElseIf !Empty(cCc) .And. !Empty(cAnexo)
                  SEND MAIL FROM cDe TO cPara CC cCc SUBJECT cAssunto BODY cMsg
                  ATTACHMENT cAnexo RESULT lResulSend
         ElseIf Empty(cCc) .And. Empty(cAnexo)
                  SEND MAIL FROM cDe TO cPara CC cCc SUBJECT cAssunto BODY cMsg
                  RESULT lResulSend
         Endif
      Endif
```

Continuação:

```
      If !lResulSend
         GET MAIL ERROR cError
         If _lJob
                ConOut(Padc("Falha no Envio do e-mail "+cError,80))
         Else
                MsgAlert("Falha no Envio do e-mail " + cError)
       Endif
      Endif

Else

   If _lJob
          ConOut(Padc("Falha na autenticação do e-mail: "+cError,80))
   Else
          MsgAlert("Falha na autenticação do e-mail:" + cError)
   Endif

Endif

DISCONNECT SMTP SERVER

IF lResulSend
   If _lJob
          ConOut(Padc("E-mail enviado com sucesso",80))
   Else
          MsgInfo("E-mail enviado com sucesso" + cError)
   Endif
ENDIF

RETURN lResulSend
```

Anotações

```
#include "protheus.ch"
#include 'tbiconn.ch'
#include "AP5MAIL.CH"

/*/
+-----------------------------------------------------------------------
| Função    | SENDMAIL      | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | ENVIO DE E-MAIL GENERICO                                  |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

USER FUNCTION FSendMail(_lJob)

Local lResulConn := .T.
Local lResulSend := .T.
Local lResult      := .T.
Local cError := ""

Local cServer  := AllTrim(GetMV("MV_RELSERV"))
Local cEmail   := AllTrim(GetMV("MV_RELACNT"))
Local cPass    := AllTrim(GetMV("MV_RELPSW"))
Local lRelauth := GetMv("MV_RELAUTH")

Local cDe      := cEmail
Local cPara    := "arnaldojr@microsiga.com.br"
Local cCc      := ""
Local cBcc     := ""
Local cAssunto := "Teste de envio de e-mail: Curso ADVPL"
Local cAnexo   := "\SYSTEM\lgrl99.bmp"
Local cMsg     := Space(200)

Default _lJob     := .F.

cMsg  := "----------------------------------------------------"
cMsg  += "CURSO DE ADVPL                                      "
cMsg  += "----------------------------------------------------"
cMsg  += "Você está recebendo um e-mail do curso de ADVPL avançado"

//CONNECT SMTP SERVER cServer ACCOUNT cEmail PASSWORD cPass RESULT lResulConn
lResulConn := MailSmtpOn( cServer, cEmail, cPass)

If !lResulConn
   //GET MAIL ERROR cError
   cError := MailGetErr()
   If _lJob
          ConOut(Padc("Falha na conexao "+cError,80))
   Else
      MsgAlert("Falha na conexao "+cError)
   Endif
   Return(.F.)
Endif
```

Continuação:

```
// Sintaxe: SEND MAIL FROM cDe TO cPara CC cCc SUBJECT cAssunto BODY cMsg
ATTACHMENT cAnexo RESULT lResulSend
// Todos os e-mail terão: De, Para, Assunto e Mensagem, porém precisa analisar
se tem: Com Cópia e/ou Anexo

If lRelauth
       lResult := MailAuth(Alltrim(cEmail), Alltrim(cPass))
   //Se nao conseguiu fazer a Autenticacao usando o E-mail completo, tenta fazer
a autenticacao usando apenas o nome de usuario do E-mail
      If !lResult
            nA := At("@",cEmail)
            cUser := If(nA>0,Subs(cEmail,1,nA-1),cEmail)
            lResult := MailAuth(Alltrim(cUser), Alltrim(cPass))
      Endif

Endif

If lResult

      /*
      If Empty(cCc) .And. Empty(cAnexo)
         SEND MAIL FROM cDe TO cPara SUBJECT cAssunto BODY cMsg RESULT
         lResulSend
      Else
         If Empty(cCc) .And. !Empty(cAnexo)
                  SEND MAIL FROM cDe TO cPara SUBJECT cAssunto BODY cMsg
                  ATTACHMENT cAnexo RESULT lResulSend
         ElseIf !Empty(cCc) .And. !Empty(cAnexo)
                  SEND MAIL FROM cDe TO cPara CC cCc SUBJECT cAssunto BODY cMsg
                  ATTACHMENT cAnexo RESULT lResulSend
         ElseIf Empty(cCc) .And. Empty(cAnexo)
                  SEND MAIL FROM cDe TO cPara CC cCc SUBJECT cAssunto BODY cMsg
                  RESULT lResulSend
         Endif
      Endif
    */

    lResulSend := MailSend(cDe,{cPara},{cCc},{cBcc},cAssunto,cMsg,{cAnexo},.T.)

      If !lResulSend
         //GET MAIL ERROR cError
         cError := MailGetErr()
         If _lJob
                ConOut(Padc("Falha no Envio do e-mail "+cError,80))
         Else
                MsgAlert("Falha no Envio do e-mail " + cError)
       Endif
      Endif

Else
   If _lJob
          ConOut(Padc("Falha na autenticação do e-mail: "+cError,80))
   Else
          MsgAlert("Falha na autenticação do e-mail:" + cError)
   Endif
Endif
```

Continuação:

```
//DISCONNECT SMTP SERVER
MailSmtpOff()

IF lResulSend
   If _lJob
          ConOut(Padc("E-mail enviado com sucesso",80))
   Else
          MsgInfo("E-mail enviado com sucesso" + cError)
   Endif
ENDIF

RETURN lResulSend
```

Anotações

Recebimento de mensagens utilizando funções

```
#include "protheus.ch"
#include "tbiconn.ch"
#include "AP5MAIL.CH"

/*/
+-----------------------------------------------------------------------
| Função    | POPMAIL       | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | RECEBIMENTO DE E-MAIL GENERICO                            |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

USER FUNCTION FPopMail(_lJob)

Local lResulConn := .T.
Local lResulPop  := .T.
Local lResult      := .T.
Local cError := ""

Local cServer  := AllTrim(GetMV("MV_RELSERV"))
Local cEmail   := AllTrim(GetMV("MV_RELACNT"))
Local cPass    := AllTrim(GetMV("MV_RELPSW"))
Local lRelauth := GetMv("MV_RELAUTH")

Local cDe      := ""
Local cPara    := ""
Local cCc      := ""
Local cBcc     := ""
Local cAssunto := ""
Local aAnexo   := {}
Local cMsg     := ""
Local cPath    := "\MailBox"
Local nMsgCount:= 0
Local nNumber  := 0

Default _lJob     := .F.

lResulConn := MailPopOn( cServer, cEmail, cPass,1000)

If !lResulConn
   cError := MailGetErr()
   If _lJob
          ConOut(Padc("Falha na conexao "+cError,80))
   Else
      MsgAlert("Falha na conexao "+cError)
   Endif
   Return(.F.)
Endif
```

Continuação:

```
/*If lRelauth
       lResult := MailAuth(Alltrim(cEmail), Alltrim(cPass))
   // Se nao conseguiu fazer a Autenticacao usando o E-mail completo, tenta
   // fazer a autenticacao usando apenas o nome de usuario do E-mail
      If !lResult
            nA := At("@",cEmail)
            cUser := If(nA>0,Subs(cEmail,1,nA-1),cEmail)
            lResult := MailAuth(Alltrim(cUser), Alltrim(cPass))
      Endif

Endif*/

If lResult

    PopMsgCount(@nMsgCount)

      For nNumber := 1 to nMsgCount

          lResulPop := MailReceive(nNumber, @cDe, @cPara, @cCc, @cBcc,;
                                   @cAssunto, @cMsg, @aAnexo , cPath, .F.)

            If !lResulPop
               cError := MailGetErr()
               If _lJob
                      ConOut(Padc("Falha no recebimento do e-mail "+cError,80))
               Else
                      MsgAlert("Falha no recebimento do e-mail " + cError)
               Endif

            Else // Salvar mensagem de e-mail

                  cMessage := "DE: "+cDe+CRLF
                  cMessage += "PARA: "+cPara+CRLF
                  cMessage += "CC: "+cCc+CRLF
                  cMessage += "BCC: "+cBcc+CRLF
                  cMessage += "ASSUNTO: "+cAssunto+CRLF
                  cMessage += "MENSAGEM: "+cMsg+CRLF

                  MsgInfo(cMessage,"E-mail Recebido")

            Endif

      End-For // Next nNumber

Else

   If _lJob
          ConOut(Padc("Falha na autenticação do e-mail: "+cError,80))
   Else
          MsgAlert("Falha na autenticação do e-mail:" + cError)
   Endif

Endif

MailPopOff()
```

```
Continuação:

IF lResulPop
   If _lJob
          ConOut(Padc("E-mails recebidos com sucesso",80))
   Else
          MsgInfo("E-mail recebidos com sucesso" + cError)
   Endif
ENDIF

RETURN lResulPop
```

43. Integração básica com MsOffice

43.1. Exportação para EXCEL

A funcionalidade básica de exportação de informações para o Microsoft Excel utiliza a função DlgToExcel(), a qual permite que as informações em formatos de array sejam geradas em uma planilha.

DLGTOEXCEL()

Realiza a exportação das informações do ambiente Protheus em formato de arrays para uma planilha do Microsoft Excel.

☑ **Sintaxe: DlgToExcel(cOrigem, cTitulo, aDadosCab, aDadosItens)**

☑ **Parâmetros:**

cOrigem	Conteúdo fixo definido como: “GETDADOS”
cTitulo	Nome para exibição da planilha
aDadosCab	Array contendo os nomes dos campos que serão exibidos na planilha.
aDadosItens	Array contendo as informações dos campos, de acordo com a ordem do array de cabeçalho.

☑ **Retorno:**

Nenhum	.

Na exportação das informações para o Microsoft Excel deve-se atentar para as colunas que possuírem informações alfanuméricas mas que contém apenas números. Para que estas colunas sejam exibidas corretamente deve ser acrescentado um caracter especial no início da string de forma que o Microsoft Excel as reconheça como texto e não como numéricas.

Exemplo de exportação para o Microsoft Excel utilizando a função DlgToExcel()

```
#include "protheus.ch"

/*/
+-----------------------------------------------------------------------
| Função    | GExpExcel     | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Descrição | EXPORTACAO PARA EXCEL GENERICO                            |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

User Function GExpExcel()

Local aCabExcel   :={}
Local aItensExcel       :={}

// AADD(aCabExcel, {"TITULO DO CAMPO", "TIPO", NTAMANHO, NDECIMAIS})
AADD(aCabExcel, {"A1_FILIAL"        ,"C", 02, 0})
AADD(aCabExcel, {"A1_COD"           ,"C", 06, 0})
AADD(aCabExcel, {"A1_LOJA"          ,"C", 02, 0})
AADD(aCabExcel, {"A1_NOME"          ,"C", 40, 0})
AADD(aCabExcel, {"A1_MCOMPRA"       ,"N", 18, 2})

MsgRun("Favor Aguardar.....", "Selecionando os Registros",;
       {|| GProcItens(aCabExcel, @aItensExcel)})

MsgRun("Favor Aguardar.....", "Exportando os Registros para o Excel",;
       {||DlgToExcel({{"GETDADOS",;
       "POSIÇÃO DE TÍTULOS DE VENDOR NO PERÍODO",;
       aCabExcel,aItensExcel}})})

Return

/*/
+-----------------------------------------------------------------------
| Função    | GProcItens    | Autor | Arnaldo R. Junior | Data |       |
+-----------------------------------------------------------------------
| Uso       | Curso ADVPL                                               |
+-----------------------------------------------------------------------
/*/

Static Function GProcItens(aHeader, aCols)

Local aItem
Local nX

DbSelectArea("SA1")
DbSetOrder(1)
DbGotop()
```

Continuação:

```
While SA1->(!EOF())

      aItem := Array(Len(aHeader))

      For nX := 1 to Len(aHeader)
             IF aHeader[nX][2] == "C"
                  aItem[nX] := CHR(160)+SA1->&(aHeader[nX][1])
             ELSE
                  aItem[nX] := SA1->&(aHeader[nX][1])
             ENDIF
      Next nX

      AADD(aCols,aItem)
      aItem := {}
      SA1->(dbSkip())

End

Return
```

Anotações

APÊNDICES

Relação de imagens para aplicações visuais

	AFASTAME		BPMSDOCI		COLTOT
	AFASTAMENTO		BPMSEDT1		CONTAINR
	ALT_CAD		BPMSEDT2		DBG05
	AMARELO		BPMSEDT3		DBG06
	ANALITICO		BPMSEDT4		DBG09
	ANALITIC		BPMSRELA		DBG3
	AGENDA		BPMSTASK1		DESTINOS
	ALTERA		BPMSTASK2		DESTINOS2
	AREA		BPMSTASK3		DISABLE
	ASIENTOS		BPMSTASK4		DISCAGEM
	AUTOM		BR_AMARELO		DOWN
	BAIXATIT		BR_AZUL		E5
	BAR		BR_AZUL_OCEAN		EDITABLE
	BMPCALEN		BR_CINZA		EXCLUIR
	BMPEMERG		BR_LARANJA		FILTRO
	BMPGROUP		BR_MARROM		FINAL
	BMPINCLUIR		BR_PRETO		FOLDER10
	BMPPERG		BR_VERDE		FOLDER11
	BMPPOST		BR_VERDE_OCEAN		FOLDER12
	BMPTABLE		BR_VERMELHO		FOLDER14
	BMPTRG		BR_VERMELHO_OCEAN		FOLDER5
	BMPUSER		BUDGET		FOLDER6
	BMPVISUAL		BUDGETY		FOLDER7
	BONUS		CADEADO		GEOROTA

	BOTTOM		CALCULADORA		GRAF2D
	BPMSDOC		CANCEL		GRAF3D
	BPMSDOCA		CHAVE2		HISTORIC
	BPMSDOCE		CHECKED		INSTRUME
	IMPRESSAO		PCO_ITALT		PMSSETATOP
	LBNO		PCO_ITEXC		PMSSETAUP
	LBOK		PCOCO		PMSTASK1
	LBTIK		PCOCUBE		PMSTASK2
	LEFT		PCOFXCANCEL		PMSTASK3
	LINE		PCOFXOK		PMSTASK4
	LIQCHECK		PENDENTE		PMSUSER
	LJPRECO		PESQUISA		PMSZOOMIN
	LOCALIZA		PGNEXT		PMSZOOMOUT
	LUPA		PGPREV		POSCLI
	MAQFOTO		PMSCOLOR		PRECO
	MATERIAL		PMSEDT3		PREV
	METAS_BAIXO_16		PMSEDT4		PRINT03
	METAS_BAIXO_LEG		PMSEXCEL		PRODUTO
	METAS_CIMA_16		PMSEXPALL		RECALC
	METAS_CIMA_LEG		PMSEXPCMP		RECORTAR
	MSGHIGH		PMSMAIS		RIGHT
	MSVISIO		PMSMATE		RPMNEW
	NEXT		PMSMENOS		RPMSAVE
	NOTE		PMSPESQ		S4SB014N
	NOVACELULA		PMSPRINT		S4WB001N
	OBJETIVO		PMSRELA		S4WB005N

	OK		PMSRRFSH		S4WB006N
	ORDEM		PMSSETABOT		S4WB007N
	PARAMETROS		PMSSETADIR		S4WB008N
	PCO_COINC		PMSSETADOWN		S4WB009N
	PCO_CONOR		PMSSETAESQ		S4WB010N
	S4WB011N		WEB		CARAGANEW
	S4WB014B		WFCHK		CARGASEQ
	S4WB016N		WFUNCHK		CCTCALC
	SALVAR		ADDCONTAINER		CHAT
	SDUIMPORT		ADICIONAR_001		CHAT1
	SDUPACK		ARMAZEM		CHAT2
	SDUPROPR		ATALHO		CHAT3
	SDUSETDEL		AVGARMAZEM		CHECK
	SDUSOFTSEEK		AVGBOX1		CHECKOK
	SHORTCUTDEL		AVGLBPAR1		CLOCK01
	SHORTCUTEDIT		AVGOIC1		CLOCK02
	SHORTCUTMINUS		AVIAO		CLOCK03
	SHORTCUTNEW		AZUL		CLOCK04
	SHORTCUTPLUS		BALANCA		DEVOLNF
	SIMULACA		BGCOLOR		COBROWSR
	SIMULACAO		BMPPARAM		COLFORM
	SUGESTAO		BMPCONS		COMPTITL
	SUMARIO		BMPCPO		COMSOM
	SVM		BMPDEL		CRITICA
	TK_VERTIT		BR_BRANCO		COPYUSER
	UNCHECKED		BRANCO		CTBLANC

	UP		BR_CANCEL		CTBREPLA
	USER		BR_MARROM		DBG07
	VCDOWN		BR_PINK		DELWEB
	VCUP		BTCALC		COLOR
	VENDEDOR		BTPESQ		DBG12
	VERNOTA		CARGA		DBG10
	DEPENDENTES		F7_VERM		F14_PINK
	GEO		F8_NULL		F14_PRET
	EDITWEB		F10_AMAR		F14_VERD
	EMPILHADEIRA		F10_AZUL		F14_VERM
	ENABLE		F10_CINZ		FÉRIAS
	ESCALA		F10_LARA		FILTRO1
	ESTOMOVI		F10_MARR		FOLDER8
	F5_AZUL		F10_NULL		FOLDER13
	F5_NULL		F10_PINK		FOLDER15
	F5_VERD		F10_PRET		FORM
	F5_VERM		F10_VERD		FRCOLOR
	F6_NULL		F10_VERM		FRTOFFLINE
	F5_AMAR		F11_NULL		FRTONLINE
	F5_CINZ		F12_AMAR		GEO
	F5_LARA		F12_AZUL		GEOEMAIL
	F5_MARR		F12_CINZ		GEOTRECHO
	F5_PINK		F12_LARA		GERPROJ
	F5_PRET		F12_MARR		GLOBO
	F7_AMAR		F12_PINK		IC_17
	F7_AZUL		F12_PRET		INSTRUME

	F7_CINZ		F12_VERD		LANDSCAPE
	F7_LARA		F12_VERM		LIGHTBLU
	F7_MARR		F14_AMAR		MDIHELP
	F7_NULL		F14_AZUL		MDILOGOFF
	F7_PINK		F14_CINZ		MDIRUN
	F7_PRET		F14_LARA		MDISPOOL
	F7_VERD		F14_MARR		MEDEXT
	MENURUN		RPMCABEC		SDUFIELDS
	MPWIZARD		RPMCPO		SDUFIND
	NCO		RPMDES		SDUGOTO
	NEWWEB		RPMFORM		SDUNEW
	NOCONNECT		RPMFUNC		SDUOPEN
	NOCHECKED		RPMGROUP		SDUOPENIDX
	NOMEDICA		RPMIMP		SDUORDER
	NORMAS		RPMIMPORT		SDURECALL
	OPEN		RPMNEW2		SDUREPL
	OPERACAO		RPMOPEN		SDUSEEK
	OUTLOOK		RPMPERG		SDUSTRUCT
	PAPEL_ESCRITO		RPMTABLE		SDUSUM
	PEDIDO		S4WB004N		SDUZAP
	PIN		S4WB013N		SEMSOM
	PMSINFO		S4WB014A		SOLICITA
	PREDIO		SALARIOS		SSFONTES
	PRINT02		SAVECLOCK		TAB1
	PROCESSA		SDUADDTBL		TABPRICE
	PRODUT2		SDUAPPEND		TEXTBOLD

	PROJETPMS		SDUCLOSE		TEXTCENTER
	PRTETQ		SDUCLOSEIDX		XCLOSE
	QMT_COND		SDUCOPYTO		TEXTITALIC
	QMT_NO		SDUCOUNT		TEXTJUSTIFY
	QMT_OK		SDUCREATEIDX		TEXTLEFT
	RESPADEX		SDUDELETE		TEXTRIGHT
	RESPONSA		SDUDRPTBL		TEXTUNDERLINE
	ROSA		SDUERASE		TK_ALTFIN
	TK_CLIFIN		BPMSEDT3A		GCT_NEW
	TK_FIND		BPMSEDT3E		INVOICE1
	TK_FONE		BPMSEDT3I		MSGGROUP
	TK_HISTORY		BPMSEDT4A		MSGHIGH
	TK_NOVO		BPMSEDT4E		PCO_COALT
	TK_REFRESH		BPMSEDT4I		PCO_COEXC
	TPOPAGTO1		BPMSREC		PCO_ITINC
	UPDWARNING		BPMSRECA		PCOCOLA
	UPDERROR		BPMSRECE		PCOCOPY
	UPDINFORMATION		BPMSRECI		PCOEDIT
	VERDE		BPMSRELAA		PCOFX
	VERMELHO		BPMSRELAE		PCOLOCK
	VERMESCURO		BPMSRELAI		PEDIDO2
	WATCH		BPMSTSK1A		PEDIDO2_MDI
	CLIENTE		BPMSTSK1E		PGRSAVE
	ACAO		BPMSTSK1I		PMSAPONT
	BOXBOM1		BPMSTSK2A		PMSCANC
	BOXBOM2		BPMSTSK2E		PMSCOLUM

	BOXBOM3		BPMSTSK2I		PMSCONS
	BOXBOM4		BPMSTSK3A		PMSCUSTO
	BOXBOM5		BPMSTSK3E		PMSDATE
	BPMSEDT1A		BPMSTSK3I		PMSESTRU
	BPMSEDT1E		BPMSTSK4A		PMSEXCEL
	BPMSEDT1I		BPMSTSK4E		PMSEXEC
	BPMSEDT2A		BPMSTSK4I		PMSEXPEXC
	BPMSEDT2E		ENGRENAGEM2		PMSFILTER
	BPMSEDT2I		GCT_EDIT		PMSGRAPH
	PMSNEXT		MSGFORWD		UNSELECTALL
	PMSOPCAO		MSGREPLY		BSTART
	PMSPESQ		OMSDIVIDE		BTURNSHARPLEFT
	PMSPREV		PMSUPDOWN		BTURNSHARPRIGHT
	PMSPREVIO		SHAPE01		ENGRENAGEM
	PMSPRINT		SHAPE02		PAGEDOWN
	PMSPROG		SHAPE03		PAGEUP
	PMSSUPALOC		SHAPE04		SELECT
	PMSTOOLS		SHAPE05		SELECTALL
	PMSUSERP		TRIDOWN		BBEARRIGHT
	REFRESH		WORD		BEND
	SHAPE06		ROTEIRO		BKEEPLEFT
	TRILEFT		CLIPS_PQ		BKEEPRIGHT
	TRIRIGHT		NEXT_PQ		BLEFT
	VINCULA1		RELACIONAMENTO_DIREIRA_PQ		BRIGHT
	BAHEAD		PREV_PQ		CANCEL
	BBEARLEFT		UP.GIF		CONFIRM

	DOWN		GOTOP		INVERTSELECTION
	GOBOTTOM				

LISTAS DE EXERCÍCIOS

Exercícios

01	Implementar a classe Aluno com os métodos New(), Inscrever() e Avaliar().

Anotações

02	Desenvolver um diálogo para interagir com a classe Aluno desenvolvida no exercício anterior.

Anotações

03	Implementar uma Enchoice para o cadastro de produtos (SB1).

Anotações

04	Converter o cadastro desenvolvido no exercício anterior para que o mesmo utilize o objeto MsMGet().

Anotações

05	Desenvolver uma interface de cadastro que combine os objetos MsMGet e MsNewGetDados causando o efeito de uma Modelo3().

Anotações

06	Adicionar na interface de cadastro desenvolvida no exercício anterior uma EnchoiceBar() com um botão definido pelo usuário que exiba o AxCadastro de um item utilizado na interface.

Anotações

07	Converter o fonte DIALOG_OBJETOS.PRW para sintaxe orientada à objetos.

Anotações

08	Desenvolver uma MaWndBrowse utilizando um arquivo temporário.

Anotações

09	Implementar uma funcionalidade com múltiplos TwBrowses que permita pesquisar clientes e visualizar os títulos a receber dos mesmos.

Anotações

10	Implementar um objeto TFolder para montar um rodapé para uma interface que contenha MsMGet() e MsNewGetDados().

Anotações

11	Implementar a MsAdvSize() e MsObjSize() no fonte desenvolvido no exercício anterior.

Anotações

12	Desenvolver uma MaWndBrowse utilizando um Alias gerado pela função TcGenQry().

Anotações

13	Desenvolver uma MaWndBrowse utilizando um Alias gerado pela função TcGenQry() para um banco de dados externo a aplicação ERP.

Anotações

14	Desenvolver uma aplicação que execute a Stored Procedure padrão de reprocessamento de saldos contábeis.

Anotações

15	Implementar no fonte WndSqlConn desenvolvido anteriormente as funções TcInclui(), TcAltera(), TcExclui() com o uso das funções TEnchoice() e TcSqlExec().

Anotações

16	Implementar uma função que realize o envio de um texto simples para uma conta de e-mail @totvs, que permita sua fácil identificação.

Anotações

17	Implementar uma rotina que realize o recebimento do e-mail enviado anteriormente para a conta de e-mail @totvs.

Anotações

18	Implementar uma rotina automática para gravar as informações no cadastro de clientes (MATA030).

Anotações

Projeto: Avaliação prática do treinamento de ADVPL Avançado

Objetivos do projeto

- ☑ Contemplar o conteúdo do curso e estimular a prática da programação utilizando a linguagem ADVPL;
- ☑ Acrescentar um grau de desafio ao treinamento;

Regras para entrega e apresentação

- ☑ Deverá ser entregue na data estabelecida pelo instrutor, contendo os fontes e demais arquivos necessários. A avaliação do projeto será realizada após a entrega, no prazo estabelecido pelo instrutor;
- ☑ Pode ser realizado em grupos de até 03 pessoas. Caso existam alunos sem grupos, estes poderão integrar equipes já formadas com o total de membros estabelecido, mediante aprovação do instrutor;

Avaliação do projeto

- ☑ Mesmo peso da prova teórica, podendo substituí-la integralmente.
- ☑ Serão considerados os quesitos:
  - ❑ Funcionalidades
  - ❑ Clareza da programação
  - ❑ Apresentação
  - ❑ Otimização de técnicas e recursos apresentados no treinamento

Funcionalidades que compõe o projeto

Desenvolver uma aplicação utilizando a linguagem ADVPL que contemple as seguintes funcionalidades:

Projeto 01	Desenvolver uma aplicação de cadastro utilizando a classe Aluno e os métodos disponíveis para a mesma.

Projeto 02	Converter os fontes do pacote de oficina de programação para orientação a objetos.

Projeto 03	Implementar uma aplicação ADVPL que consulte e atualize dados em um banco de interface externo ao Protheus.

REFERÊNCIAS BIBLIOGRÁFICAS

Referências bibliográficas

Gestão empresarial com ERP
Ernesto Haberkorn, 2006

Programação Orientada a Objetos com C ++
Ivan Luiz Marques Ricarte, 1996

Modelagem e Projetos baseados em objetos
James Rumbaugh, 1994 – 11ª. Edição

Programação orientada à objetos no FiveWin
Gilmer – FiveWin Brasil

Oficina de Programação
Robson Luiz Estefani Gonçalves

Apostila de Treinamento - TReport
Tânia Bronzeri

Apostila de Treinamento - ADVPL
Educação corporativa

Apostila de Treinamento – Boas Práticas de Programação
Inteligência Protheus e Fábrica de Software

DEM – Documentação Eletrônica Microsiga
Microsiga Software S.A.

Materiais diversos de colaboradores Microsiga
Colaboradores Microsiga

Colaboradores

Revisão 01	Data: 04.2008
Arnaldo Raymundo Junior	CSA
Luis Akira Tamura	Educação Corporativa
Márcia Satiko Sasaki Tokura	ACR-N1_FRAMEW1
Patricia Lopes Legas	Educação Corporativa
Robson Luiz Estefani Gonçalves	Fábrica de Software
Sérgio Sueo Fuzinaka	Inteligência Protheus