



Educação Corporativa

Programação ADVPL Intermediário



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

ESTRUTURA DO TREINAMENTO

OBJETIVOS DO CURSO.....	11
MÓDULO 04: Desenvolvendo aplicações em ADVPL.....	12
1. A linguagem ADVPL.....	12
2. Estrutura de um Programa ADVPL.....	14
2.1. Áreas de um Programa ADVPL.....	16
3. Declaração e Atribuição de Variáveis.....	19
3.1. Tipo de Dados.....	19
3.2. Declaração de variáveis.....	20
3.3. Escopo de variáveis.....	21
3.4. Entendendo a influência do escopo das variáveis.....	25
4. Regras adicionais da linguagem ADVPL.....	26
4.1. Palavras reservadas.....	26
4.2. Pictures de formatação disponíveis.....	27
5. Programas de Atualização.....	28
5.1. Modelo1() ou AxCadastro().....	29
5.2. Mbrowse().....	32
5.2.1. AxFunctions().....	37
5.2.2. FilBrowse().....	40
5.2.3. EndFilBrw().....	40
5.2.4. PesqBrw().....	41
5.2.5. BrwLegenda().....	41
5.3. MarkBrowse().....	45
5.3.1. Funções de Apoio.....	46
5.4. Modelo2().....	49
5.4.1. Componentes de uma tela no formato Modelo 2.....	49
5.4.2. Estrutura de um programa utilizando a Modelo2().....	58
5.4.3. Função Modelo2().....	66
5.5. Modelo3().....	71
5.5.1. Estrutura de um programa utilizando a Modelo3().....	74
5.5.2. Função Modelo3().....	83
6. Arquivos e Índices Temporários.....	87
6.1. Utilização de arquivos e índices temporários.....	87
6.2. Funções para manipulação de arquivos e índices temporários.....	87
6.2.1. CriaTrab().....	87
6.2.2. dbUseArea().....	88
6.2.3. IndRegua().....	88
6.3. Funções Auxiliares para Arquivos de Trabalho e Temporários.....	89
7. Relatórios não gráficos.....	93
7.1. Funções Utilizadas para Desenvolvimento de Relatórios.....	93
7.1.1. SetPrint().....	93
7.1.2. SetDefault().....	94

7.1.3.	RptStatus()	95
7.1.3.1.	SETREGUA()	95
7.1.3.2.	INCREGUA()	96
7.1.4.	CABEC()	96
7.1.5.	RODA()	97
7.1.6.	Pergunte()	97
7.1.7.	AjustaSX1()	97
7.1.8.	PutSX1()	99
7.2.	Estrutura de Relatórios Baseados na SetPrint()	101
8.	Manipulação de arquivos I.....	111
8.1.	Geração e leitura de arquivos em formato texto.....	111
8.1.1.	1ª Família de funções de gravação e leitura de arquivos texto	112
8.1.1.1.	FCREATE()	112
8.1.1.2.	FWRITE()	113
8.1.1.3.	FCLOSE()	113
8.1.1.4.	FSEEK()	114
8.1.1.5.	FOPEN()	114
8.1.1.6.	FREAD()	115
8.1.2.	2ª Família de funções de gravação e leitura de arquivos texto	122
8.1.2.1.	FT_FUSE()	122
8.1.2.2.	FT_FGOTOP()	122
8.1.2.3.	FT_FLASTREC()	122
8.1.2.4.	FT_FEOF()	123
8.1.2.5.	FT_FREADLN()	123
8.1.2.6.	FT_FSKIP()	123
8.1.2.7.	FT_FGOTO()	124
8.1.2.8.	FT_FRECNO()	124
9.	Oficina de programação I	127
9.1.	Interfaces com sintaxe clássica.....	127
9.2.	Réguas de processamento.....	131
9.2.1.	RptStatus()	131
	SETREGUA()	133
	INCREGUA()	134
9.2.2.	Processa()	135
	SETPROC()	137
	INCPROC()	137
9.2.3.	MsNewProcess()	138
9.2.4.	MsAguarde()	140
9.2.5.	MsgRun()	142
9.3.	ListBox()	143
9.3.1.	Listbox simples.....	143
9.3.2.	Listbox múltiplas colunas	146
9.4.	ScrollBar()	148
9.5.	ParamBox().....	152
MÓDULO 05: Introdução a orientação à objetos.....		157
10.	Conceitos de orientação à objetos	157
10.1.	Definições	157
10.2.	Conceitos Básicos	160
10.3.	O Modelo de Objetos (OMT)	161
10.3.1.	Objetos e Classes.....	161
10.3.2.	Atributos	162

10.3.3.	Operações e Métodos	163
10.3.4.	Sugestões de desenvolvimento	164
11.	Orientação a objetos em ADVPL	165
11.1.	Sintaxe e operadores para orientação a objetos	165
11.2.	Estrutura de uma classe de objetos em ADVPL	167
11.3.	Implementação dos métodos de uma classe em ADVPL	168
MÓDULO 06:	ADVPL Orientado à objetos I.....	172
12.	Componentes da interface visual do ADVPL	172
12.1.	Particularidades dos componentes visuais.....	179
12.1.1.	Configurando as cores para os componentes	179
13.	Aplicações com a interface visual do ADVPL	181
13.1.	Captura de informações simples (Multi-Gets)	181
13.1.1.	Enchoice()	182
13.1.2.	MsMGet()	184
13.2.	Captura de múltiplas informações (Multi-Lines).....	186
13.2.1.	MsGetDB()	187
13.2.2.	MsGetDados()	191
13.2.3.	MsNewGetDados().....	195
13.2.3.1.	Definindo cores personalizadas para o objeto MsNewGetDados()	200
13.3.	Barras de botões.....	205
13.3.1.	EnchoiceBar().....	205
13.3.2.	TBar()	207
13.3.3.	ButtonBar	208
13.3.4.	Imagens pré-definidas para as barras de botões.....	211
APÊNDICES.....		212
BOAS PRÁTICAS DE PROGRAMAÇÃO.....		212
14.	Arredondamento	212
15.	Utilização de Identação	212
16.	Capitulação de Palavras-Chave	214
16.1.	Palavras em maiúsculo	214
17.	Utilização da Notação Húngara.....	215
18.	Técnicas de programação eficiente.....	215
GUIA DE REFERÊNCIA RÁPIDA: Funções e Comandos ADVPL.....		227
Conversão entre tipos de dados		227
CTOD()		227
CVALTOCHAR().....		227
DTOD()		228
DTOS()		228
STOD()		228
STR()		229
STRZERO().....		229
VAL().....		230
Matemáticas.....		231
ACOS()		231
CEILING().....		231
COS()		231
LOG10().....		232

SIN()	232
SQRT()	233
TAN()	233
Análise de variáveis.....	234
TYPE()	234
VALTYPE()	234
Manipulação de arrays.....	235
AADD()	235
ACLONE()	236
ACOPY()	236
ADEL()	237
ADIR()	238
AFILL()	239
AINS()	239
ARRAY()	240
ASCAN()	240
ASCANX()	241
ASIZE()	242
ASORT()	243
ATAIL()	244
Manipulação de blocos de código	245
EVAL()	245
DBEVAL()	245
AEVAL()	247
Manipulação de strings.....	248
ALLTRIM()	248
ASC()	248
AT()	249
BITON()	250
CAPITAL()	250
CHR()	250
DESCEND()	251
GETDTOVAL()	251
ISALPHA()	252
ISDIGIT()	252
ISLOWER()	253
ISUPPER()	253
LEN()	253
LOWER()	254
LTRIM()	254
MATHC()	255
OEMTOANSI()	255
PADL() / PADR() / PADC()	256
RAT()	256
REPLICATE()	257
RTRIM()	257
SPACE()	258
STRTOKARR()	258
STRTRAN()	259
STUFF()	259
SUBSTR()	260
TRANSFORM()	260
UPPER()	261
Manipulação de data / hora.....	262
CDOW()	262
CMONTH()	262

DATE()	263
DAY()	263
DOW()	264
DTOC()	264
DTOS()	265
ELAPTIME()	265
MONTH()	266
SECONDS()	266
TIME()	267
YEAR()	267
Manipulação de variáveis numéricas	268
ABS()	268
ALEATORIO()	268
INT()	269
NOROUND()	269
RANDOMIZE()	270
ROUND()	270
Manipulação de arquivos	271
ADIR()	271
CGETFILE()	272
CPYS2T()	278
CPYT2S()	278
CURDIR()	279
DIRECTORY()	280
DIRREMOVE()	281
DISKSPACE()	282
EXISTDIR()	283
FCLOSE()	283
FCREATE()	284
FERASE()	285
FILE()	285
FILENOEXT()	286
FOPEN()	287
FREAD()	289
FREADSTR ()	289
FRENAME()	290
FSEEK()	291
FT_FEOF()	291
FT_FGOTO()	292
FT_FGOTOP()	292
FT_FLASTREC()	292
FT_FREADLN()	293
FT_FRECNO()	294
FT_FSKIP()	294
FT_FUSE()	294
FWRITE()	295
MSCOPYFILE()	297
MSCOPYTO()	298
MSCREATE()	298
MSERASE()	299
MSRENAME()	300
RETFILENAME()	300
Manipulação de arquivos e índices temporários	301
CRIATRAB()	301
Manipulação de bases de dados	302
ALIAS()	302
BOF() / EOF()	302

COPY()	303
COPY STRUCTURE()	306
DBAPPEND()	306
DBCLEARALLFILTER()	307
DBCLEARFILTER()	308
DBCLEARINDEX()	308
DBCLOSEALL()	309
DBCLOSEAREA()	309
DBCOMMIT()	310
DBCOMMITALL()	310
DBCREATE()	311
DBCREATEINDEX()	312
DBDELETE()	313
DBF()	314
DBFIELDINFO()	314
DBFILTER()	315
DBGOTO()	316
DBGOTOP()	316
DBGOBOTTON()	317
DBINFO()	317
DBNICKINDEXKEY()	318
DBORDERINFO()	319
DBORDERNICKNAME()	320
DBPACK()	320
DBRECALL()	321
DBRECORDINFO()	321
DBREINDEX()	322
DBRLOCK()	323
DBRLOCKLIST()	323
DBRUNLOCK()	324
DBSETDRIVER()	324
DBSETINDEX()	325
DBSETNICKNAME()	326
DBSELECTAREA()	326
DBSETORDER()	327
DBSEEK() E MSSEEK()	328
DBSKIP()	329
DBSETFILTER()	330
DBSTRUCT()	331
DBUNLOCK()	331
DBUNLOCKALL()	332
DBUSEAREA()	332
DELETED()	333
FCOUNT()	333
FOUND()	334
INDEXKEY()	334
INDEXORD()	335
LUPDATE()	335
MSAPPEND()	336
MSUNLOCK()	336
ORDBAGEXT()	337
ORDKEY()	337
RECLOCK()	338
RECNO()	339
SELECT()	339
SET FILTER TO	340
SOFTLOCK()	341
USED()	342
ZAP	342

Controle de numeração sequencial.....	343
GETSXENUM()	343
CONFIRMSXE()	343
ROLLBACKSXE()	343
Validação	344
ALLWAYSFALSE()	344
ALLWAYSTRUE()	344
EXISTCHAV()	344
EXISTCPO()	345
LETTERORNUM()	345
NAOVAZIO()	345
NEGATIVO()	345
PERTENCE()	346
POSITIVO()	346
TEXTO()	346
VAZIO()	346
Manipulação de parâmetros do sistema.....	347
GETMV()	347
GETNEWPAR()	347
PUTMV()	348
SUPERGETMV()	348
Controle de impressão	349
AVALIMP()	349
CABEC()	350
IMPCADAST()	353
MS_FLUSH()	353
OURSPOOL()	355
RODA()	356
SETDEFAULT()	358
SETPRC()	359
SETPRINT()	359
Controle de processamentos.....	361
ABREEXCL()	361
CLOSEOPEN()	361
CLOSESFILE()	361
CHKFILE()	362
CONOUT()	363
CRIAVAR()	363
DISARMTRANSACTION()	364
EXECBLOCK()	365
EXISTBLOCK()	365
ERRORBLOCK()	366
FINAL()	368
FINDFUNCTION()	368
FUNDESC()	369
FUNNAME()	369
GETAREA()	369
GETCOUNTRYLIST()	370
ISINCALLSTACK()	370
REGTOMEMORY()	371
RESTAREA()	372
USEREXCEPTION()	372
Utilização de recursos do ambiente ERP	374
AJUSTASX1()	374
ALLUSERS()	376
ALLGROUPS()	378

CGC()	379
CONPAD1()	379
DATAVALIDA()	380
EXISTINI()	380
EXTENSO()	381
FORMULA()	381
GETADVFFVAL()	381
HELP()	382
MESEXTENSO()	383
OBRIGATORIO()	383
OPENFILE()	386
PERGUNTE()	387
PESQPICT()	387
PESQPICTQT()	388
POSICIONE()	388
PUTSX1()	389
RETINDEX()	390
SIXDESCRICAO()	390
TABELA()	391
TAMXSX3()	391
TM()	392
X1DEF01()	393
X1PERGUNT()	394
X2NOME()	394
X3CBOX()	395
X3DESCRIC()	395
X3PICTURE()	396
X3TITULO()	397
X3USO()	397
X5DESCRI()	398
X6CONTEUD()	399
X6DESCRIC()	400
XADDESCRIC()	401
XBDESCRI()	401
XFILIAL()	402
Componentes da interface visual	403
MSDIALOG()	403
MSGET()	404
SAY()	405
BUTTON()	405
SBUTTON()	406
CHECKBOX()	406
COMBOBOX()	408
FOLDER()	409
RADIO()	410
Interfaces de cadastro	411
AXCADASTRO()	411
MBROWSE()	411
AXPESQUI()	411
AXVISUAL()	412
AXINCLUI()	412
AXALTERA()	413
AXDELETA()	414
Interfaces visuais para aplicações	415
ALERT()	415
AVISO()	415
FORMBACTH()	416

MSGFUNCTIONS()	417
Recursos das interfaces visuais	418
GDFIELDGET()	418
GDFIELDPOS()	418
GDFIELDPUT()	418
GETMARK()	419
MARKBREFRESH()	420
READVAR()	420

OBJETIVOS DO CURSO

Objetivos específicos do curso:

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

a) Conceitos a serem aprendidos

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

b) Habilidades e técnicas a serem aprendidas

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

c) Atitudes a serem desenvolvidas

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

MÓDULO 04: Desenvolvendo aplicações em ADVPL

1. 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.

2. 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.

2.1. Áreas de um Programa ADVPL

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

```
#include protoheus.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
```

3. Declaração e Atribuição de Variáveis

3.1. 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.

3.2. 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.

3.3. 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 mantém seu valor atual quando a execução da função Pai termina. Entretanto, somente pode ser acessada por execuções subsequentes da função Pai.

Variáveis de escopo private

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

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

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

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

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

Por exemplo:

```
Function Pai()  
Private nVar := 10  
<comandos>  
.   
Filha()  
<mais comandos>  
.   
Return(.T.)
```

Neste exemplo, a variável nVar é criada com escopo private e inicializada com o valor 10. Quando a função Filha é executada, nVar ainda existe e, 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.



Importante

No ambiente ERP Protheus, existe uma convenção adicional a qual deve ser respeitada que variáveis em uso pela aplicação não sejam incorretamente manipuladas. Por esta convenção deve ser adicionado o 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 privadas, nVar ainda existe após o término da execução da função Pai.

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



Importante

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

Exemplo: Public _cRotina



Anotações

3.4. 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. Regras adicionais da linguagem ADVPL

4.1. 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			



Importante

- ☑ 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 "_".

4.2. 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

Al_NOME - Character - 40
Picture: @!

5. 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).

The screenshots illustrate the three models of data entry in Protheus:

- Modelo 1 (AxCadastro):** A full-screen form for registering a client, with fields for various attributes like name, address, and contact information.
- Modelo 2:** A form for a purchase order, featuring a table with columns for item, product, unit, quantity, unit price, and total value.
- Modelo 3:** A form for a sales order, similar to the purchase order but with a different layout for the item table and summary section.

5.1. 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
```

Exercício

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

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado

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
```

Exercício

Implementar no AxCadastro as validações de alteração e exclusão.

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado

5.2. 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: MBrowse(nLin1, nCol1, nLin2, nCol2, cAlias, aFixe, cCpo, nPar08, cFun, nClickDef, aColors, cTopFun, cBotFun, nPar14, bInitBloc, INoMnuFilter, ISeeAll, IChgAll)**

☑ **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.
bl nitBloc	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.
INoMnuFilter	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.
ISeeAll	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.
ICHgAll	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 ISeeAll 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	<p>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:</p> <p>[n][1] - Título; [n][2] - Rotina; [n][3] - Reservado; [n][4] - Operação (1 - pesquisa; 2 - visualização; 3 - inclusão; 4 - alteração; 5 - exclusão);</p> <p>Ele ainda pode ser parametrizado com as funções básicas da AxCadastro conforme abaixo:</p> <p>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})</p>
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.



Importante

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, convencionalmente 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" ,"AxAlterar" ,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 específica 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
```

5.2.1. 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, IMaximized)
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, IF3,; cTransact, aButtons, aParam, aAuto, IVirtual, IMaximized)
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, IF3,; cTransact, aButtons, aParam, aAuto, IVirtual, IMaximized)
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, IMaximized)
Descrição	Função de exclusão padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

Exercício

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

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado



Anotações

5.2.2. FilBrowse()

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

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

☑ **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()
IShowProc	Habilita (.T.) ou desabilita (.F.) a apresentação da mensagem "Selecionando registros ...", no processamento.

5.2.3. 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

5.2.4. 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, IShowProc) }

5.2.5. 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"}}



Importante

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

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

AADD(aRotina,{"Pesquisar" , "PesqBrw" ,0,1})
AADD(aRotina,{"Visualizar" , "AxVisual" ,0,2})
AADD(aRotina,{"Incluir" , "U_BInclui" ,0,3})
AADD(aRotina,{"Alterar" , "U_BAlterar" ,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(bFiltrarBrw)  
  
dbSelectArea(cAlias)  
dbGoTop()  
mBrowse(6,1,22,75,cAlias,,,,,aCores)  
  
//+-----  
//| Deleta o filtro utilizado na função FilBrowse  
//+-----  
EndFilBrw(cAlias,aIndexSA2)  
  
Return Nil  
  
//+-----  
//| Função: BInclui - Rotina de Inclusão  
//+-----  
User Function BInclui(cAlias,nReg,nOpc)  
  
Local nOpcao := 0  
  
nOpcao := AxInclui(cAlias,nReg,nOpc)  
  
If nOpcao == 1  
    MsgInfo("Inclusão efetuada com sucesso!")  
Else  
    MsgInfo("Inclusão cancelada!")  
Endif  
  
Return Nil  
  
//+-----  
//| Função: BAltera - Rotina de Alteração  
//+-----  
User Function BAltera(cAlias,nReg,nOpc)  
  
Local nOpcao := 0  
  
nOpcao := AxAltera(cAlias,nReg,nOpc)  
  
If nOpcao == 1  
    MsgInfo("Alteração efetuada com sucesso!")  
Else  
    MsgInfo("Alteração cancelada!")  
Endif  
  
Return Nil
```

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
```

Exercício

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

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado

Formatado: Fonte: Itálico,
Sem sublinhado



Anotações

5.3. 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.

5.3.1. Funções de Apoio

GetMark: define a marca atual.

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

ThisMark: captura a marca em uso.

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

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

Exemplo: Função MarkBrow() e 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"              , "AxAlterar" , 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])
```



```

Next nX

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

```

Exercício

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

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado

Formatado: Fonte: Itálico,
Sem sublinhado

Exercício

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

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado

Formatado: Fonte: Itálico,
Sem sublinhado

Exercício

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

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado

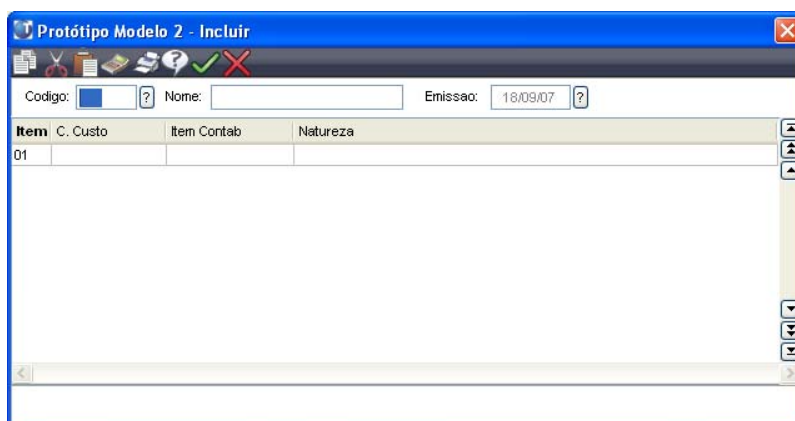
Formatado: Fonte: Itálico,
Sem sublinhado

clientes selecionados

5.4. 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:

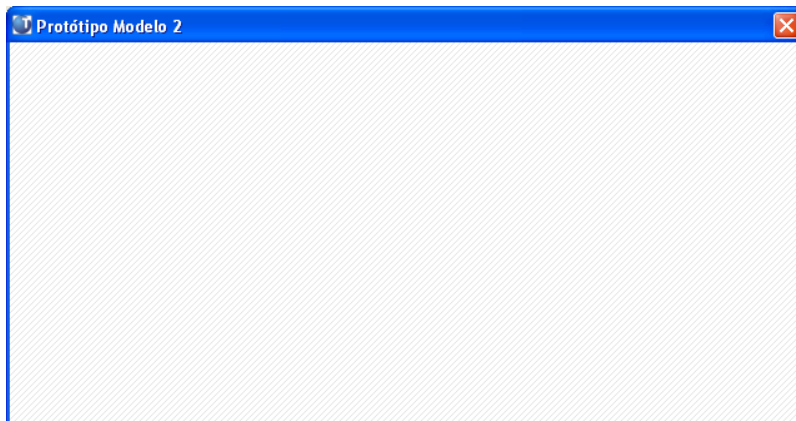
- ❑ **Função Modelo2()** – Trata-se de uma função pronta que contempla o protótipo Modelo 2, porém, este é um assunto que não iremos tratar aqui, visto que é uma funcionalidade simples que quando necessário intervir em algo na rotina não há muito recurso para tal.
- ❑ **Protótipo Modelo 2** – Trata-se 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.



5.4.1. 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*, [*IMensApag*] , [*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.
IMensApag	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}.

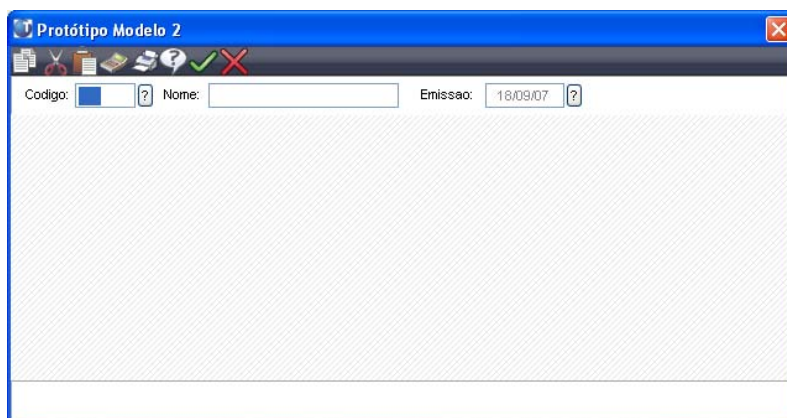


Figura: Protótipo Modelo2 – Enchoice

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], [IPar6], [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.
ICentered	Exibe o texto do título centralizado.
IPar6	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.
ILowered	Exibe o painel rebaixado em relação ao controle de fundo.
IRaised	Exibe a borda do controle rebaixado em relação ao controle de fundo.

Comando SAY - Objeto: TSay()

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

☒ Sintaxe SAY:

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

☒ Sintaxe TSay(): TSay():New([anRow], [anCol], [abText], [aoWnd], [acPicture], [aoFont], [IPar7], [IPar8], [IPar9], [alPixels], [anClrText], [anClrBack], [anWidth], [anHeight], [IPar15], [IPar16], [IPar17], [IPar18], [IPar19])

☒ Parâmetros:

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.
IPar7	Reservado.
IPar8	Reservado.
IPar9	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.
IPar15	Reservado.
IPar16	Reservado.
IPar17	Reservado.
IPar18	Reservado.
IPar19	Reservado.

Comando MSGET - Objeto: TGet()

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

☒ Sintaxe MSGET:

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

☒ Sintaxe TGet():New([anRow], [anCol], [abSetGet], [aoWnd], [anWidth], [anHeight], [acPict], [abValid], [anClrFore], [anClrBack], [aoFont], [IPar12], [oPar13], [alPixel], [cPar15], [IPar16], [abWhen], [IPar18], [IPar19], [abChange], [alReadOnly], [alPassword], [cPar23], [acReadVar], [cPar25], [IPar26], [nPar27], [IPar28])

☒ Parâmetros:

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.
IPar12	Reservado.
oPar13	Reservado.
alPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
cPar15	Reservado.
IPar16	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.
IPar18	Reservado.

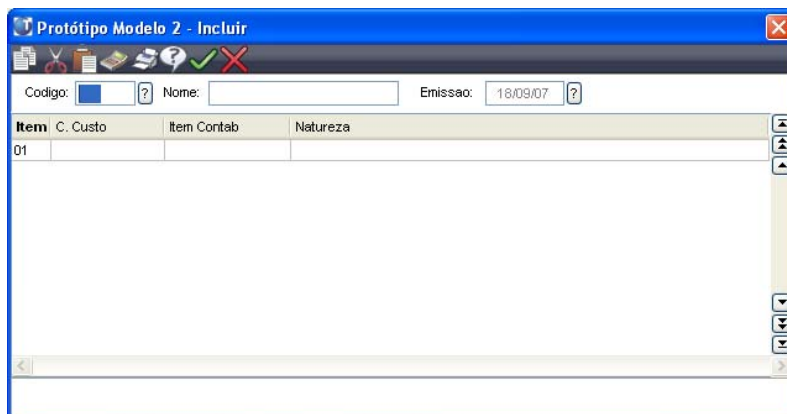
IPar19	Reservado.
abChange	Bloco de código, opcional. Executado quando o controle modifica o valor da variável associada.
alReadOnly	Lógico, opcional. Se .T. o controle não poderá ser editado.
alPassword	Lógico, opcional. Se .T. o controle exibirá asteriscos "*" no lugar dos caracteres exibidos pelo 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.
IPar26	Reservado.
nPar27	Reservado.
IPar18	Reservado.

Objeto MsGetDados()

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

5.4.2. 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

    oTPanel := TPanel():New(0,0,"",oDlg,NIL,.T.,;
        .F.,NIL,NIL,0,16,.T.,.F.)
    oTPanel: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() ) )
        EndWhile
    EndIf
```

```

        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

```

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

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

5.4.3. Função Modelo2()

A função Modelo2() é uma interface pré-definida pela Microsiga que implementa de forma padronizada os componentes 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()



Importante

- ☐ 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, "AlwaysTrue()", ;
               " ", "C", " ", "R"})
AADD(aHeader,{ "Descricao" , "X5_DESCRI", "@!", 40, 0, "AlwaysTrue()", ;
               " ", "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
```

Exercício

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

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado

Formatado: Fonte: Itálico,
Sem sublinhado

Exercício

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

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado

Formatado: Fonte: Itálico,
Sem sublinhado

Exercício

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

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado

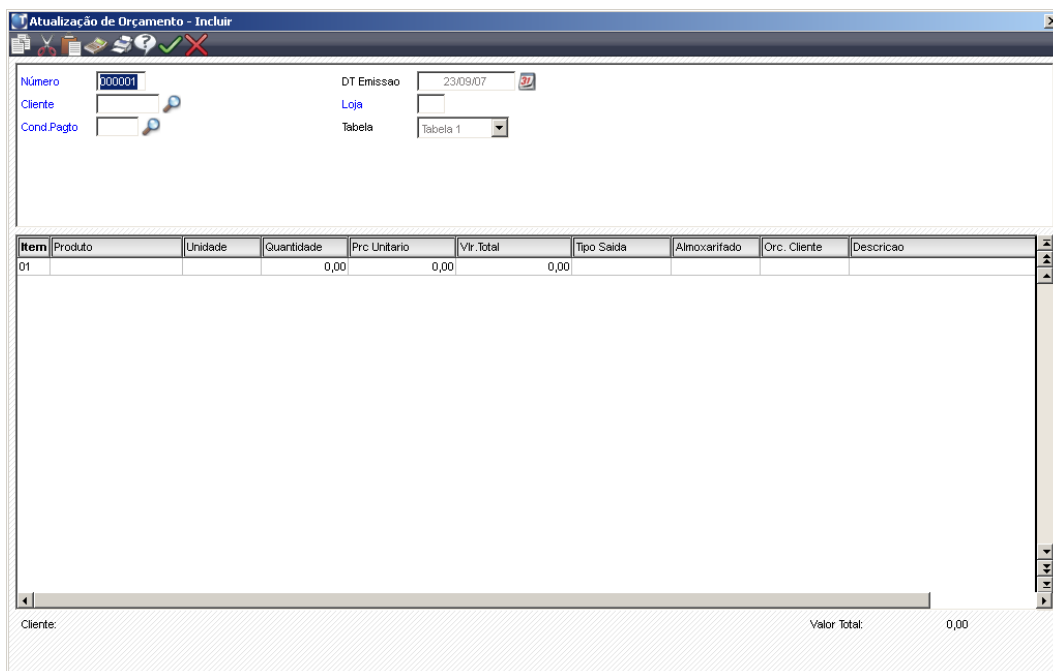
Formatado: Fonte: Itálico,
Sem sublinhado

5.5. Modelo3()

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

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

Para ganharmos tempo não será 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:



Item	Produto	Unidade	Quantidade	Prc Unitario	Vir. Total	Tipo Saída	Almoxarifado	Orc. Cliente	Descricao
01			0,00	0,00	0,00				

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 esta 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

5.5.1. 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
```

Função de Inclusão

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

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

dbSelectArea( cAlias )
dbSetOrder(1)

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

Mod3aHeader()
Mod3aCOLS( nOpc )

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

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

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

    oGet := 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
```

Função de Visualização, Alteração e Exclusão

```
//+-----+
//| Rotina | Mod3Mnt | Autor | Robson Luiz (rleg) | Data | 01.01.2007 |
//+-----+
//| Descr. | Rotina para Visualizar, Alterar e Excluir dados. |
//+-----+
//| Uso    | Para treinamento e capacitação. |
//+-----+
User Function Mod3Mnt( cAlias, nReg, nOpc )
Local oDlg
Local oGet
Local nX := 0
Local nOpcA := 0
Private aHeader := {}
Private aCOLS := {}
Private aGets := {}
Private aTela := {}
Private aREG := {}

dbSelectArea( cAlias )
dbSetOrder(1)

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

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

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

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

    U_Mod3Cli()

    oGet := 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),
"Al_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, aAlterar )
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
```

5.5.2. Função Modelo3()

A função Modelo3() é uma interface pré-definida pela Microsiga que implementa de forma padronizada os componentes 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 realizada 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], [cTudoOk], [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,{ "Alterar" ,"U_Mod3All"     ,0,4})
AADD(aRotina,{ "Exclui"  ,"U_Mod3All"     ,0,5})

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

Return

User Function Mod3All(cAlias,nReg,nOpcx)

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

Exemplo (continuação):

```
//ÚAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA;
//³ Opcoes de acesso para a Modelo 3
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAU
Do Case
    Case nOpcx==3; nOpce:=3 ; nOpcG:=3 // 3 - "INCLUIR"
    Case nOpcx==4; nOpce:=3 ; nOpcG:=3 // 4 - "ALTERAR"
    Case nOpcx==2; nOpce:=2 ; nOpcG:=2 // 2 - "VISUALIZAR"
    Case nOpcx==5; nOpce:=2 ; nOpcG:=2 // 5 - "EXCLUIR"
EndCase

//ÚAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA;
//³ Cria variaveis M->???? da Enchoice
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAU
RegToMemory("SC5",(nOpcx==3 .or. nOpcx==4 )) // Se for inclusao ou alteracao
permite alterar o conteudo das variaveis de memoria

//ÚAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA;
//³ Cria aHeader e aCols da GetDados
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAU
nUsado:=0
dbSelectArea("SX3")
dbSeek("SC6")
aHeader:={}
While !Eof().And.(x3_arquivo=="SC6")
    If Alltrim(x3_campo)!="C6_ITEM"
        dbSkip()
        Loop
    Endif
    If X3USO(x3_usado).And.cNivel>=x3_nivel
        nUsado:=nUsado+1
        Add(aHeader,{ TRIM(x3_titulo), x3_campo, x3_picture,;
            x3_tamanho, x3_decimal,"AlwaysTrue()",;
            x3_usado, x3_tipo, x3_arquivo, x3_context } )
    Endif
    dbSkip()
End

If nOpcx==3 // Incluir
    aCols:={}Array(nUsado+1)}
    aCols[1,nUsado+1]:=F.
    For nX:=1 to nUsado
        aCols[1,nX]:=Criavar(aHeader[nX,2])
    Next
Else
    aCols:={}
    dbSelectArea("SC6")
    dbSetOrder(1)
    dbSeek(xFilial()+M->C5_NUM)
    While !eof().and.C6_NUM==M->C5_NUM
        AADD(aCols,Array(nUsado+1))
        For nX:=1 to nUsado
            aCols[Len(aCols),nX]:=FieldGet(FieldPos(aHeader[nX,2]))
        Next
        aCols[Len(aCols),nUsado+1]:=F.
        dbSkip()
    End
Endif
```

[illegible]

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

Formatado: Fonte: Itálico,
Sem sublinhado

6. Arquivos e Índices Temporários

6.1. 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.

6.2. Funções para manipulação de arquivos e índices temporários

6.2.1. CriaTrab()

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

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

☑ **Parâmetros**

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



Importante

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



Anotações

6.2.2. 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(**INewArea**, **cDriver**, **cName**, **cAlias**, **IShared**, **IReadOnly**)

☑ **Parâmetros**

INewArea	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 esteve aberto
IShared	A tabela terá acesso exclusivo ou compartilhado
IReadOnly	Se verdadeiro a tabela será somente leitura

6.2.3. 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**, **IShow**)

☑ **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
IShow	Apresentar a tela de progresso do índice/filtro temporário



Anotações

6.3. Funções Auxiliares para Arquivos de Trabalho e Temporários

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

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

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

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

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

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

Exemplo 01: Geração de arquivo e índice temporários

```
#include "protheus.ch"

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

User Function GeraTrab()

Local aStru      := {}
Local aArqTRB    := {}
Local 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, { "ALIQUIPI"    , "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

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

cNomArq := CriaTrab( aArqTRB, .T. )

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

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

// ( ... ) fazer o processamento necessário

dbSelectArea( "TRB" )
dbCloseArea()

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

Return Nil

```



Importante

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



Dica

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

Exemplo 02: Utilizando dois índices temporários com RDD DBFCDX

```
/*/  
+-----+  
| Programa | IndTwoReg | Autor | MICHEL DANTAS | Data | |  
+-----+  
| Desc. | Utilização de dois índices temporários com DBFCDX | |  
+-----+  
| Uso | Curso de ADVPL | |  
+-----+  
/*/  
  
User Function IndTwoReg()  
  
LOCAL nOrder := 0  
LOCAL cArq1 := CriaTrab(NIL,.F.)  
LOCAL cChave1 := "A1_FILIAL+A1_EST"  
LOCAL cArq2 := CriaTrab(NIL,.F.)  
LOCAL cChave2 := "A1_FILIAL+A1_NOME"  
  
dbSelectArea("SA1")  
IndRegua("SA1",cArq1,cChave1,,,"Selecionando Regs...")  
IndRegua("SA1",cArq2,cChave2,,,"Selecionando Regs...")  
  
nOrder := RetIndex("SA1")  
  
dbSetIndex(cArq1+OrdBagExt())  
dbSetIndex(cArq2+OrdBagExt())  
  
Alert("Agora vai por estado")  
  
dbsetOrder(nOrder+1)  
dbGoTop()  
While !Eof()  
    Alert("Estado : " + SA1->A1_EST + " "+" Nome : " + SA1->A1_NOME)  
    dbSkip()  
End  
  
Alert("Agora vai por nome")  
  
dbSetOrder(nOrder+2)  
dbGoTop()  
While !Eof()  
    Alert("Estado : " + SA1->A1_EST + " "+" Nome : " + SA1->A1_NOME)  
    dbSkip()  
End  
  
RetIndex("SA1")  
Ferase(cArq1+OrdBagext())  
Ferase(cArq2+OrdBagext())  
  
Return
```

Exercício

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

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado

Formatado: Fonte: Itálico,
Sem sublinhado

Exercício

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

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado

Formatado: Fonte: Itálico,
Sem sublinhado

7. 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.

7.1. Funções Utilizadas para Desenvolvimento de Relatórios

7.1.1. SetPrint()

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

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

☑ **Parâmetros:**

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

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

☒ **Retorno:**

Character	Nome do Relatório
------------------	-------------------

☒ **Estrutura aReturn:**

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

7.1.2. SetDefault()

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

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

☒ **Parâmetros:**

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

☑ **Estrutura aReturn:**

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

7.1.3. RptStatus()

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

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

☑ **Parâmetros:**

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.

7.1.3.1. SETREGUA()

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

☑ **Sintaxe: SetRegua(nMaxProc)**

☑ **Parâmetros:**

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

7.1.3.2. 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.
------------------	--

7.1.4. CABEC()

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

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

☑ **Parâmetros:**

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

☑ **Parâmetros (continuação):**

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

7.1.5. RODA()

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

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

☑ **Parâmetros:**

uPar01	Não é mais utilizado
uPar02	Não é mais utilizado
cSize	Tamanho do relatório ("P", "M", "G")

7.1.6. Pergunte()

A função PERGUNTE() inicializa as variáveis de pergunta (mv_par01,...) baseada na pergunta cadastrado no Dicionário de Dados (SX1). Se o parâmetro IAsk não for especificado ou for verdadeiro será exibida a tela para edição da pergunta e se o usuário confirmar as variáveis serão atualizadas e a pergunta no SX1 também será atualizada.

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

☑ **Parâmetros:**

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

☑ **Retorno:**

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

7.1.7. AjustaSX1()

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

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

☑ **Parâmetros:**

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

☑ Estrutura – Item do array aPerg:

Posição	Campo	Tipo	Descrição
01	X1_PERGUNT	Caractere	Descrição da pergunta em português
02	X1_PERSPA	Caractere	Descrição da pergunta em espanhol
03	X1_PERENG	Caractere	Descrição da pergunta em inglês
04	X1_VARIAVL	Caractere	Nome da variável de controle auxiliar (mv_ch)
05	X1_TIPO	Caractere	Tipo do parâmetro
06	X1_TAMANHO	Numérico	Tamanho do conteúdo do parâmetro
07	X1_DECIMAL	Numérico	Número de decimais para conteúdos numéricos
08	X1_PRESEL	Numérico	Define qual opção do combo é a padrão para o parâmetro.
09	X1_GSC	Caractere	Define se a pergunta será do tipo G – Get ou C – Choice (combo)
10	X1_VALID	Caractere	Expressão de validação do parâmetro
11	X1_VAR01	Caractere	Nome da variável MV_PAR+“Ordem” do parâmetro
12	X1_DEF01	Caractere	Descrição da opção 1 do combo em português
13	X1_DEFSPA1	Caractere	Descrição da opção 1 do combo em espanhol
14	X1_DEFENG1	Caractere	Descrição da opção 1 do combo em inglês
15	X1_CNT01	Caractere	Conteúdo padrão ou ultimo conteúdo definido como respostas para a pergunta.
16	X1_VAR02	Caractere	Não é informado
17	X1_DEF02	Caractere	Descrição da opção X do combo em português
18	X1_DEFSPA2	Caractere	Descrição da opção X do combo em espanhol
19	X1_DEFENG2	Caractere	Descrição da opção X do combo em inglês
20	X1_CNT02	Caractere	Não é informado
21	X1_VAR03	Caractere	Não é informado
22	X1_DEF03	Caractere	Descrição da opção X do combo em português
23	X1_DEFSPA3	Caractere	Descrição da opção X do combo em espanhol
24	X1_DEFENG3	Caractere	Descrição da opção X do combo em inglês
25	X1_CNT03	Caractere	Não é informado
26	X1_VAR04	Caractere	Não é informado
27	X1_DEF04	Caractere	Descrição da opção X do combo em português
28	X1_DEFSPA4	Caractere	Descrição da opção X do combo em espanhol
29	X1_DEFENG4	Caractere	Descrição da opção X do combo em inglês
30	X1_CNT04	Caractere	Não é informado
31	X1_VAR05	Caractere	Não é informado
32	X1_DEF05	Caractere	Descrição da opção X do combo em português
33	X1_DEFSPA5	Caractere	Descrição da opção X do combo em espanhol
34	X1_DEFENG5	Caractere	Descrição da opção X do combo em inglês
35	X1_CNT05	Caractere	Não é informado
36	X1_F3	Caractere	Código da consulta F3 vinculada ao parâmetro
37	X1_GRPXSG	Caractere	Código do grupo de campos SXG para atualização automática, quando o grupo for alterado.

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

7.1.8. PutSX1()

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

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

☑ **Parâmetros:**

cGrupo	Grupo de perguntas do SX1 (X1_GRUPO)
cOrdem	Ordem do parâmetro no grupo (X1_ORDEM)
cPergunt	Descrição da pergunta em português
cPerSpa	Descrição da pergunta em espanhol
cPerEng	Descrição da pergunta em inglês
cVar	Nome da variável de controle auxiliar (X1_VARIAVL)
cTipo	Tipo do parâmetro
nTamanho	Tamanho do conteúdo do parâmetro
nDecimal	Número de decimais para conteúdos numéricos
nPresel	Define qual opção do combo é a padrão para o parâmetro.
cGSC	Define se a pergunta será do tipo G – Get ou C – Choice (combo)
cValid	Expressão de validação do parâmetro
cF3	Código da consulta F3 vinculada ao parâmetro
cGrpSxg	Código do grupo de campos SXG para atualização automática, quando o grupo for alterado.
cPyme	Se a pergunta estará disponível no ambiente Pyme

cVar01	Nome da variável MV_PAR+"Ordem" do parâmetro.
cDef01	Descrição da opção 1 do combo em português
cDefSpa1	Descrição da opção 1 do combo em espanhol
cDefEng1	Descrição da opção 1 do combo em inglês
cCnt01	Conteúdo padrão ou ultimo conteúdo definido como respostas para este item
cDef0x	Descrição da opção X do combo em português
cDefSpax	Descrição da opção X do combo em espanhol
cDefEngx	Descrição da opção X do combo em inglês
aHelpPor	Vetor simples contendo as linhas de help em português para o parâmetro.
aHelpEng	Vetor simples contendo as linhas de help em inglês para o parâmetro.
aHelpSpa	Vetor simples contendo as linhas de help em espanhol para o parâmetro.
cHelp	Conteúdo do campo X1_HELP

7.2. 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]]

#IFDEF 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, "
cQuery += "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

#IFDEF 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,i
cSeq,i
aP[i,1],aP[i,1],aP[i,1],i
cMvCh,i
aP[i,2],i
aP[i,3],i
aP[i,4],i
0,i
aP[i,5],i
aP[i,6],i
aP[i,7],i
" ",i
" ",i
cMvPar,i
aP[i,8],aP[i,8],aP[i,8],i
" ",i
aP[i,9],aP[i,9],aP[i,9],i
aP[i,10],aP[i,10],aP[i,10],i
aP[i,11],aP[i,11],aP[i,11],i
aP[i,12],aP[i,12],aP[i,12],i
aHelp[i],i
{} ,i
{} ,i
" ")
Next i

Return

```

Exercício

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

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado

Formatado: Fonte: Itálico,
Sem sublinhado

8. Manipulação de arquivos I

8.1. 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().



Importante

A diferença entre as duas famílias, está na leitura do arquivo texto. Quando se tratar de arquivo texto com tamanho fixo das linhas, poderão ser utilizadas as duas famílias para leitura do arquivo, porém, quando se tratar de arquivo texto com tamanho variável das linhas, somente poderá ser utilizada a segunda família, representada pelas funções: FT_FUse(), FT_FGoTo(), FT_FRecno(), FT_FGoTop(), FT_FLastRec(), FT_FEof(), FT_FReadLn() e FT_FSkip().

8.1.1. 1ª Família de funções de gravação e leitura de arquivos texto

8.1.1.1. FCREATE()

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

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

☒ Atributos definidos no include FileIO.ch

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

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

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

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

☒ Parâmetros:

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

☒ Retorno:

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

8.1.1.2. FWRITE()

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

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

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

☑ **Parâmetros:**

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

☑ **Retorno:**

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

8.1.1.3. FCLOSE()

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

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

☑ **Sintaxe:** FCLOSE (< nHandle >)

☑ **Parâmetros:**

nHandle	Handle do arquivo obtido previamente através de FOPEN() ou FCREATE().
----------------	---

☑ **Retorno:**

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

8.1.1.4. FSEEK()

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

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

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

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

☑ **Parâmetros:**

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

☑ **Retorno:**

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

8.1.1.5. FOPEN()

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

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

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

☑ **Parâmetros:**

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

☑ **Retorno:**

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

8.1.1.6. FREAD()

Função que realiza a leitura dos dados a partir um arquivo aberto, através de FOPEN(), FCREATE() e/ou FOPENPORT(), e armazena os dados lidos por referência no buffer informado. FREAD() lerá até o número de bytes informado em nQtdBytes; caso aconteça algum erro ou o arquivo chegue ao final, FREAD() retornará um número menor que o especificado em nQtdBytes. FREAD() lê normalmente caracteres de controle (ASC 128, ASC 0, etc.) e lê a partir da posição atual do ponteiro atual do arquivo , que pode ser ajustado ou modificado pelas funções FSEEK() , FWRITE() ou FREADSTR().

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

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

☑ **Parâmetros:**

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

☑ **Retorno:**

Númerico	Quantidades de bytes lidos. Caso a quantidade seja menor que a solicitada, isto indica erro de leitura ou final de arquivo, Verifique a função FERROR() para maiores detalhes.
-----------------	--

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)->Al_FILIAL  
cLin += (cAlias)->Al_COD  
cLin += (cAlias)->Al_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 utilizadas 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

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

//1234567890123456789012345678901234567890123456789012345678901234567890
//00000000010000000002000000000300000000040000000005000000000600000000070
//FFCCCCCLLNNNNNNNNNNNNNNNNNNNNNNVVVVVVVVVVVVVVVVDDDDDDDD
//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
```

8.1.2. 2ª Família de funções de gravação e leitura de arquivos texto

8.1.2.1. FT_FUSE()

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

☑ **Sintaxe:** FT_FUSE ([cTXTFile])

☑ **Parâmetros:**

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

☑ **Retorno:**

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

8.1.2.2. FT_FGOTOP()

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

☑ **Sintaxe:** FT_FGOTO (< nPos >)

☑ **Parâmetros:**

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

8.1.2.3. FT_FLASTREC()

Função que retorna o número total de linhas do arquivo texto aberto pela FT_FUse. As linhas são delimitadas pela sequência de caracteres CRLF o LF.

☑ **Sintaxe:** FT_FLASTREC ()

☑ **Parâmetros:**

Nenhum	.
---------------	---

☑ **Retorno:**

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

8.1.2.4. FT_FEOF()

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

☒ **Sintaxe:** FT_FEOF()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Lógico	Retorna true caso o ponteiro do arquivo tenha chegado ao final, false caso contrário.
--------	---

8.1.2.5. FT_FREADLN()

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

☒ **Sintaxe:** FT_FREADLN()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

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

8.1.2.6. FT_FSKIP()

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

☒ **Sintaxe:** FT_FSKIP ([nLinhas])

☒ **Parâmetros:**

nLinhas	nLinhas corresponde ao número de linhas do arquivo TXT ref. movimentação do ponteiro de leitura do arquivo.
---------	---

☒ **Retorno**

Nenhum	.
--------	---

8.1.2.7. FT_FGOTO()

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

☒ **Sintaxe:** FT_FGOTO (< nPos >)

☒ **Parâmetros:**

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

☒ **Retorno:**

Nenhum	.
---------------	---

8.1.2.8. FT_FRECNO()

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

A função FT_FRecno retorna a posição corrente do ponteiro do arquivo texto aberto pela FT_FUse.

☒ **Sintaxe:** FT_FRECNO ()

☒ **Parâmetros:**

Nenhum	.
---------------	---

☒ **Retorno:**

Caracter	Retorna a posição corrente do ponteiro do arquivo texto.
-----------------	--

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

```
#Include "Protheus.ch"

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

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

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

```

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

FormBatch( cCadastro, aSay, aButton )

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

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

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

/****
*
* cGetFile(<ExpC1>,<ExpC2>,<ExpN1>,<ExpC3>,<ExpL1>,<ExpN2>)
* -----
* <ExpC1> - Expressão de filtro
* <ExpC2> - Título da janela
* <ExpN1> - Número 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

```

Exemplo (continuação):

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

Exercício

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

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado

Formatado: Fonte: Itálico,
Sem sublinhado

Exercício

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

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado

Formatado: Fonte: Itálico,
Sem sublinhado

9. Oficina de programação I

9.1. 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
```



Importante

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()

Executar o fonte DIALOG_OBJETOS.PRW e avaliar a definição dos componentes utilizados utilizando a sintaxe clássica.

Formatado: Centralizado

Formatado: Fonte: Itálico,
Sem sublinhado

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 VARIABEL PROMPT cTexto WHEN WHEN UNIDADE OF oObjetoRef SIZE nLargura,nAltura MESSAGE cMensagem
Descrição	Define o componente visual CheckBox, o qual permite a utilização de 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 VARIABEL 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 VARIABEL 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.

9.2. Régua de processamento

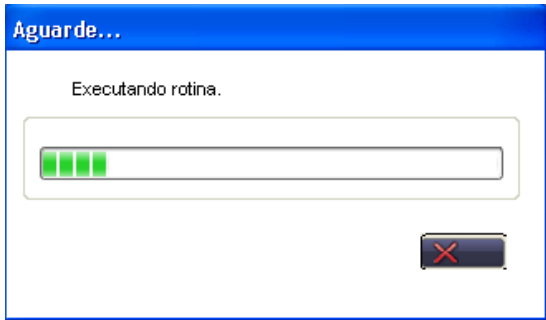
Os indicadores de progresso ou régua de processamento disponíveis na linguagem ADVPL que serão abordados neste material são:

- ❏ RPTSTATUS()
- ❏ PROCESSA()
- ❏ MSNEWPROCESS()
- ❏ MSAGUARDE()
- ❏ MSGRUN()

9.2.1. 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**


9.2.2. 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 PROCSSA() e acessórios

```
/*/  
+-----+-----+-----+-----+-----+-----+  
| Função   | GPROCES1 | Autor | ROBSON LUIZ | Data | |  
+-----+-----+-----+-----+-----+-----+  
| Descrição | Programa que demonstra a utilização das funções PROCSSA() |  
|           | 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   | PROCSSA() |
+-----+-----+-----+-----+-----+
| 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
...

```

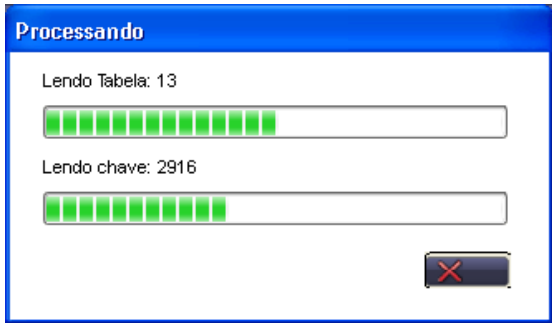
9.2.3. MsNewProcess().

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
```

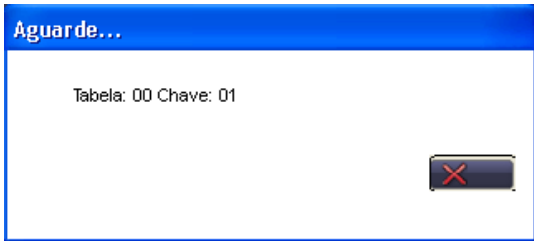
Exemplo (continuação):

```
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
```

9.2.4. 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	

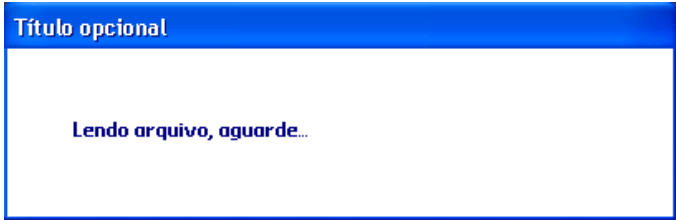
Exemplo: MSAGuarde()

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

9.2.5. 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
```

9.3. 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).

9.3.1. 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

```

9.3.2. 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.

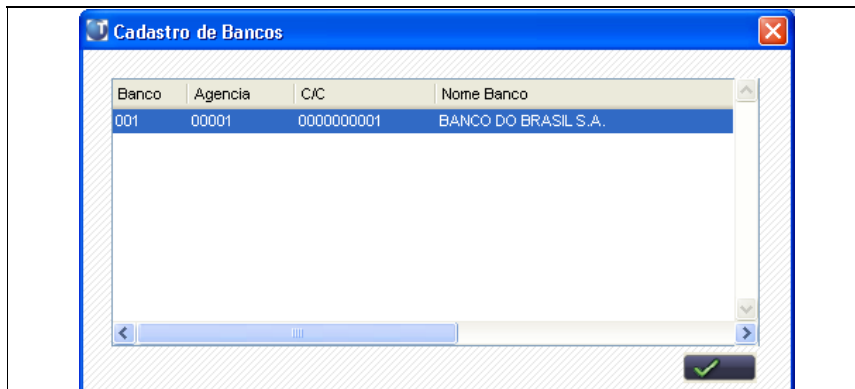
☑ **Métodos:**

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
```

9.4. 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:

Parcelamento

Processo: P00001
Prefixo: UNI
Tipo: MAN
Cliente: 000001
Loja: 01
Nome: JOSE DA SILVA SANTOS SOARES
CNPJ/CPF: 00.000.000/0001-91
Dt.Processo: 26/03/03
Total R\$: 5922
Total UFESP: 1000
Status: Z

Títulos

Parcela	Vencdo	Vencdo.Real	V
A	26/03/03	26/03/03	1.0
A	26/03/03	26/03/03	1.0
A	26/03/03	26/03/03	1.0

AIIPM

AIIPM	Placa	Data Multa
1234	DCD9815	26/03/03
1234	DCD9815	26/03/03
1234	DCD9815	26/03/03

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

9.5. 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.

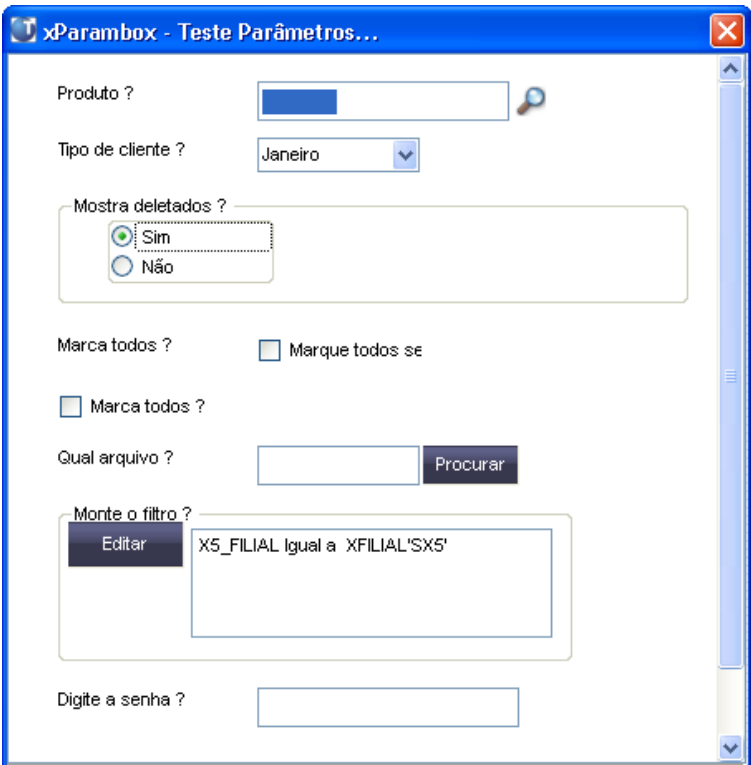
Cada componente da ParamBox será associado a um parâmetro Private denominado MV_PARxx, de acordo com a ordem do componente na tela. Os parâmetros da ParamBox podem ser utilizados de forma independente em uma rotina específica, ou complementando opções de uma rotina padrão.

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: IOK → 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 XPBOX(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

10. 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**

10.1. 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.

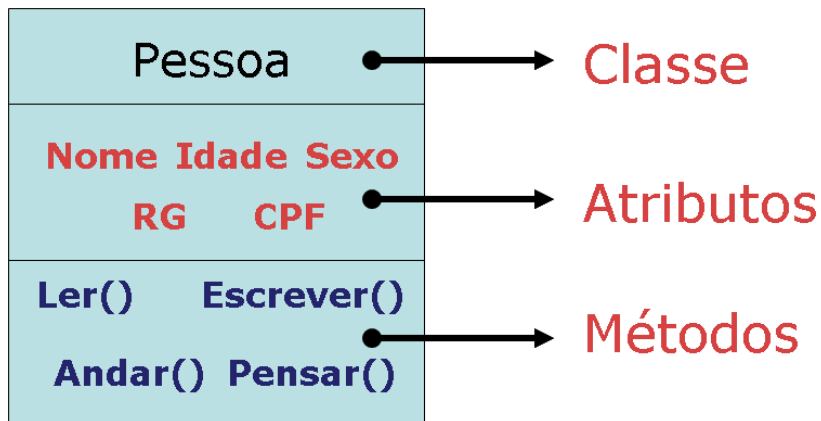


Figura: Representação de uma classe de objetos

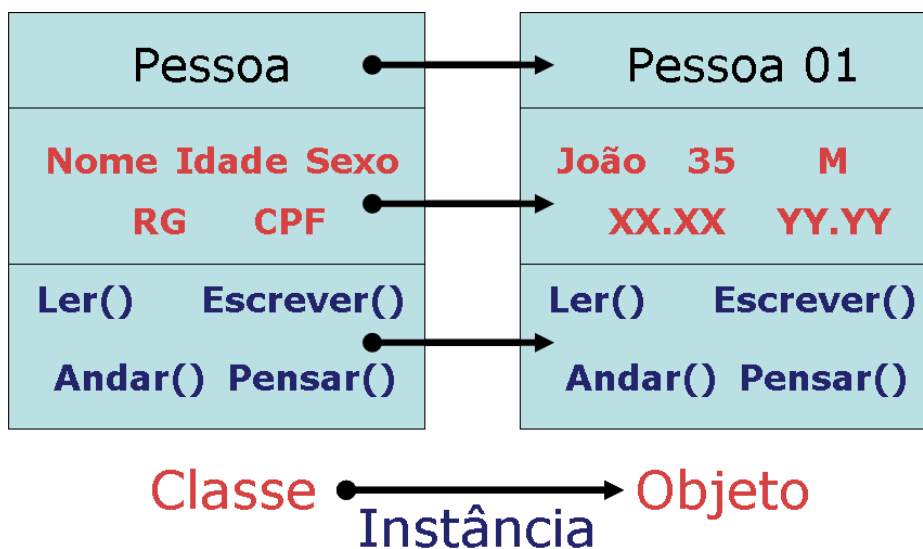


Figura: Representação de um objeto instanciado de uma classe

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, ou 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.

10.2. 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.

10.3. 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.

10.3.1. 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.

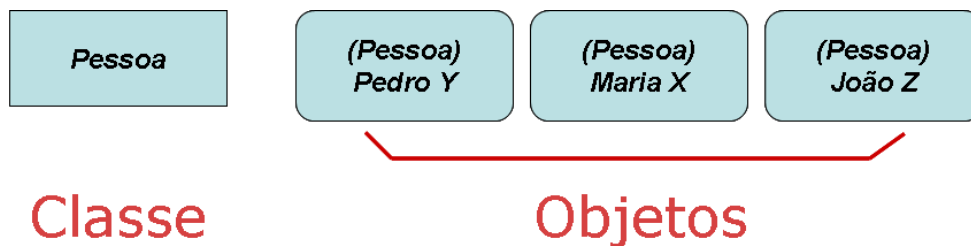


Figura: Representação diagramática de OMT para classes e objetos

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.

10.3.2. 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.

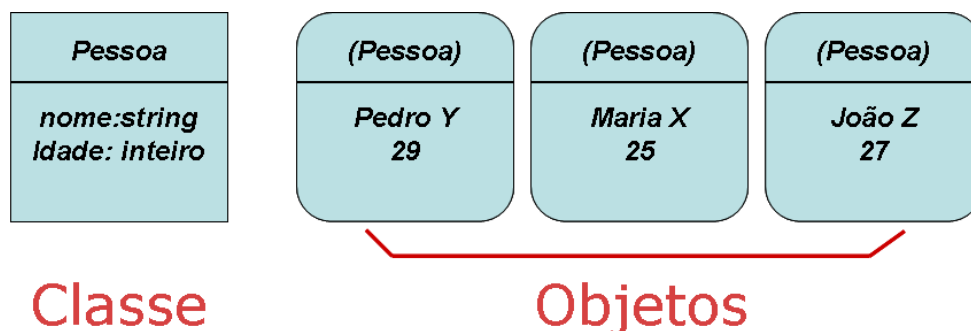


Figura: Representação diagramática de OMT para classes e objetos com atributos

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.

10.3.3. 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:

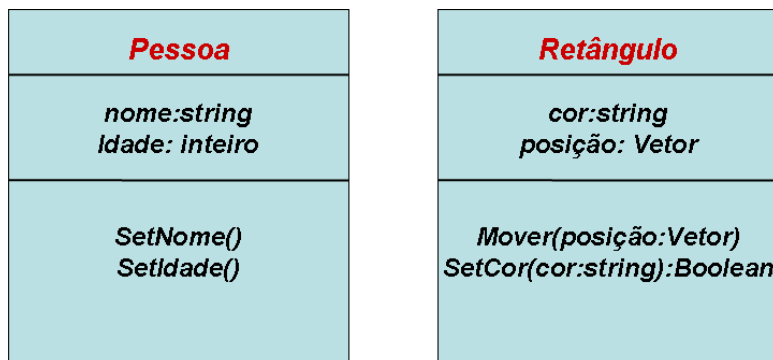


Figura: Representação diagramática de OMT para classes com atributos e operações

10.3.4. 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.

11. 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.

11.1. 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	<pre>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</pre>

11.2. 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
```

11.3. 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 princípio da orientação a objetos, a linguagem ADVPL permite que uma classe receba por herança os métodos e atributos definidos em uma outra classe, a qual tornasse a superclasse desta instância.

Para utilizar este recurso deve ser utilizada a palavra reservada "FROM" na declaração da classe, especificando a superclasse que será referenciada.

Em ADVPL o exemplo prático desta situação é a superclasse TSrvObject, a qual é utilizada pela maioria das classes e componentes da interface visual, como demonstrado no módulo 06.

Exemplo - parte 01: Declaração da classe Pessoa

```
#include "protheus.ch"
CLASS Pessoa()

DATA cNome
DATA nIdade
DATA dNascimento

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

ENDCLASS
```

Exemplo - parte 02: Declaração da classe Aluno

```
#include "protheus.ch"
CLASS Aluno() FROM Pessoa

DATA nID
DATA aCursos

METHOD Create() CONSTRUCTOR
METHOD Inscrever()
METHOD Avaliar()
METHOD GetNota()
METHOD GetStatus()

ENDCLASS

// Os objetos da classe Aluno, possuem todos os métodos e atributos da classe
Pessoa, além
// dos métodos e atributos declarados na própria classe.
```

Construtor para classes com herança

Quanto é utilizado o recurso de herança entre classes, o construtor da classe instanciada deve receber um tratamento adicional, para que o objeto instanciado seja criado com os atributos e métodos definidos na superclasse.

Nestes casos, logo após a definição do método construtor da classe, deverá ser executado o método construtor da superclasse.

Exemplo - parte 03: Método Construtor da classe Aluno

```
METHOD Create(cNome,dNascimento,nID)
:Create(cNome,dNascimento) // Chamada do método construtor da classe Pessoa.

::nID := ID

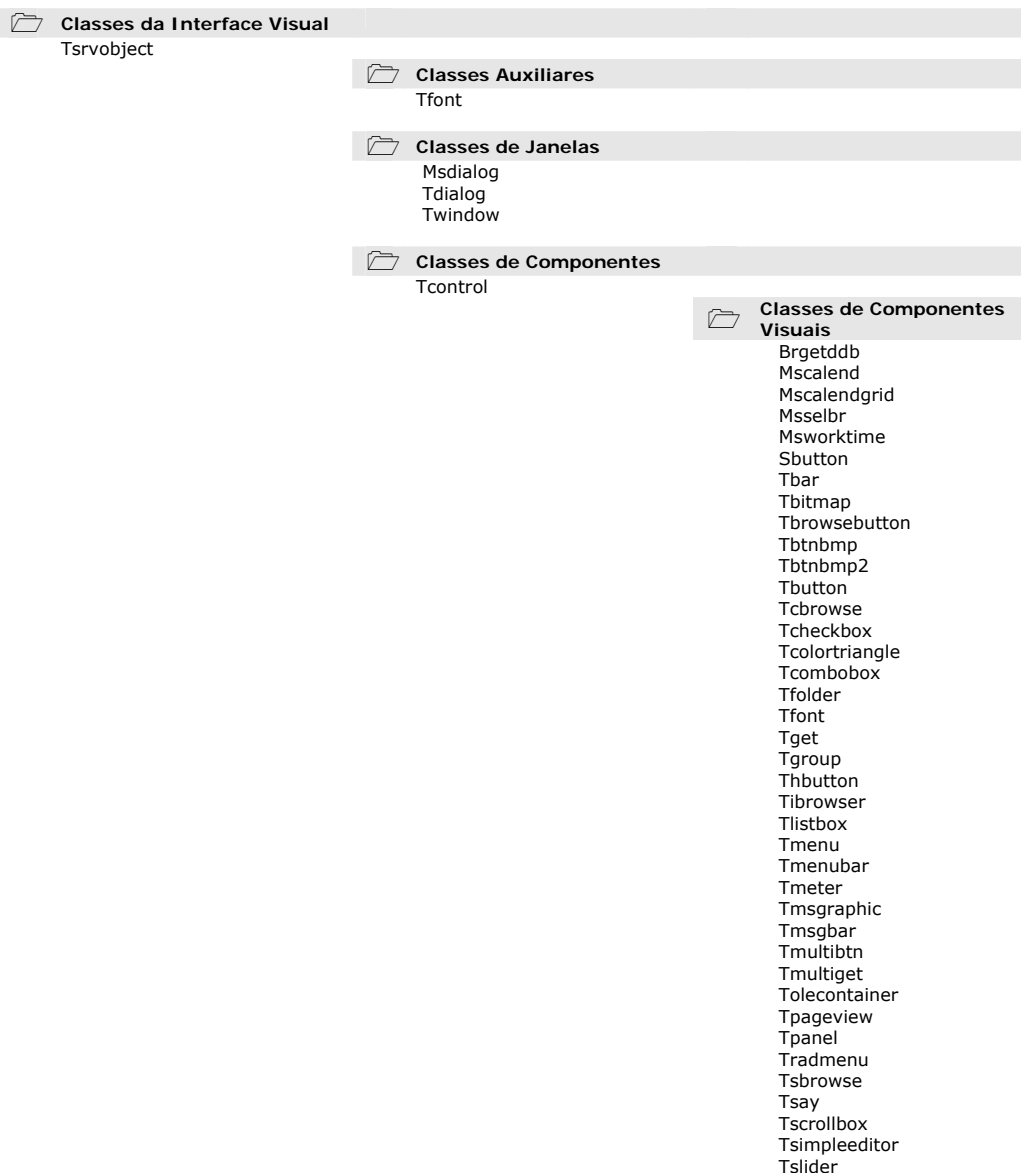
Return SELF
```

MÓDULO 06: ADVPL Orientado à objetos I

Neste módulo serão tratados os componentes e objetos da interface visual da linguagem ADVPL, permitindo o desenvolvimento de aplicações com interfaces gráficas com sintaxe orientada a objetos.

12. 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()

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.
-----------	---

TMSGGRAPHIC()

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 – tScrollbar, 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.
-----------	--

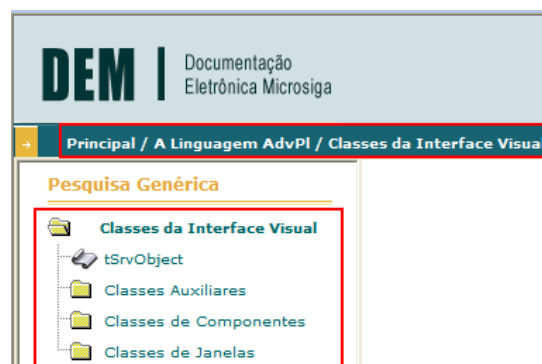
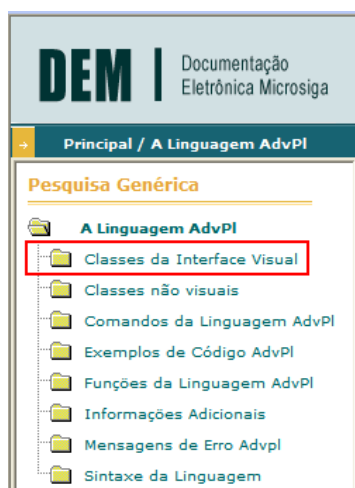
VCBROWSE()

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:



12.1.Particularidades dos componentes visuais

12.1.1. 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:

$$nCor := nVermelho + (nVerde * 256) + (nAzul * 65536)$$

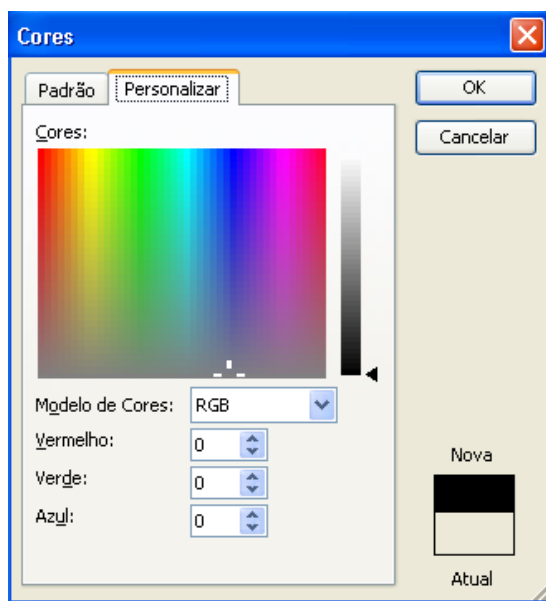


Figura: Paleta de cores no formato RGB

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

13. 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**

13.1. 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"):

A imagem mostra uma janela de software intitulada "Clientes - Incluir". No topo, há uma barra de menu com ícones para Copiar, Recortar, Colar, Calo, Spool, Ajuda, Geo, Crédito, OK e Cancelar. Abaixo, há uma barra de abas com "Cadastrais", "Adm/Fin.", "Fiscais", "Vendas" e "Outros". O formulário principal contém campos para: Código, Loja, Física/Jurid (menu suspenso), Nome, N Fantasia, Tipo (menu suspenso), Endereço, Município, Estado (menu suspenso com lupa), Bairro, CEP, DDI, DDD, Telefone, Telex, FAX, País, CNPJ/CPF, Contato, Ins. Estad., RG/Ced.Estr., Ins. Municip., DLiber/Nasc, E-Mail, Home-Page, Cod.CBD, Cod.CHRE, Bloqueado (menu suspenso com "Não" selecionado) e Insc.Rural.

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().

13.1.1. Enchoice()

- ☑ **Sintaxe:** Enchoice(cAlias, nReg, nOpc, aCRA, cLetra, cTexto, aAcho, aPos, aCpos, nModelo, nColMens, cMensagem, cTudoOk, oWnd, IF3, IMemoria, IColumn, caTela, INoFolder, IProperty)

- ☑ **Retorno:** Nil
- ☑ **Parâmetros:**

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.
IF3	Indica se a enchoice esta sendo criada em uma consulta F3 para utilizar variáveis de memória
IMemoria	Indica se a enchoice utilizará variáveis de memória ou os campos da tabela na edição
IColumn	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
INoFolder	Indica se a enchoice não irá utilizar as Pastas de Cadastro (SXA)
IProperty	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
```

13.1.2. 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

IMemoria	Indica se a enchoice utilizará variáveis de memória ou os campos da tabela na edição
IColumn	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
INoFolder	Indica se a enchoice não irá utilizar as Pastas de Cadastro (SXA)
IProperty	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   | MBRWMSGGET | 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 $ "Al_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
```

13.2. 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()**
- ☐ **MAWNCDBROWSE()**
- ☐ **MBROWSE()**

Neste tópico serão tratadas as grids digitáveis disponíveis na linguagem ADVPL para o desenvolvimento de interfaces de cadastros e manutenção de informações.

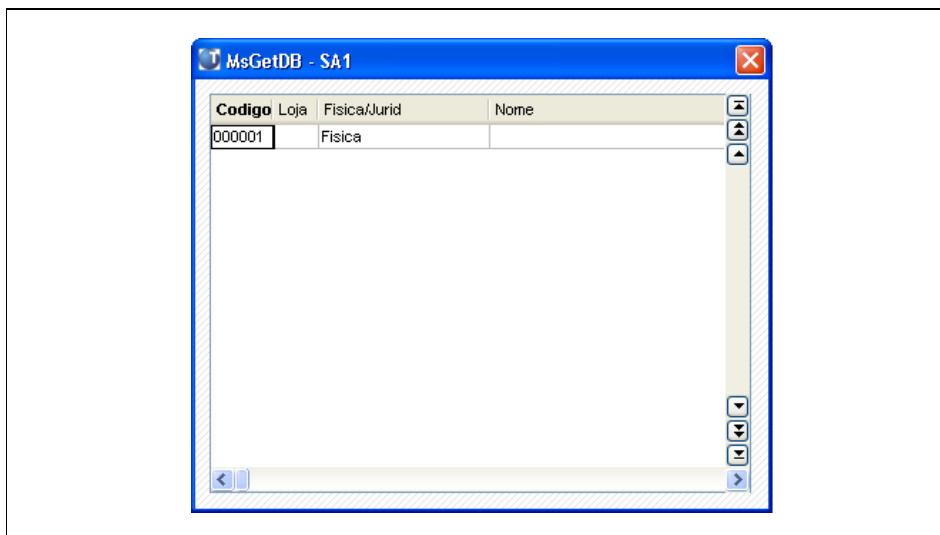
13.2.1. MsGetDB()

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	<p>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:</p> <p>{cTitulo, cRotina, nOpção, nAcesso}, aonde:</p> <p>nOpção segue o padrão do ERP Protheus para:</p> <ol style="list-style-type: none"> 1- Pesquisar 2- Visualizar 3- Incluir 4- Alterar 5- Excluir
aHeader	<p>Vetor com informações das colunas no formato:</p> <p>{cTitulo, cCampo, cPicture, nTamanho, nDecimais,; cValidação, cReservado, cTipo, xReservado1, xReservado2}</p> <p>A tabela temporária utilizada pela MsGetDB deverá ser criada com base no aHeader mais um último campo tipo lógico que determina se a linha foi excluída.</p>
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:lcCentered := .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.
```

13.2.2. MsGetDados()

A classe de objetos visuais MsGetDados() permite a criação de um grid digitável com uma ou mais colunas, baseado em um array.

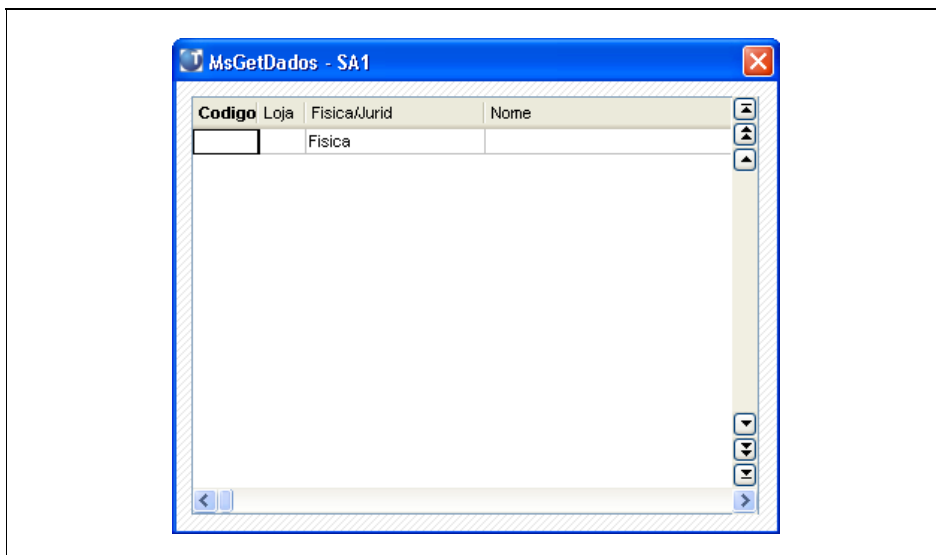
☑ **Sintaxe:** MsGetDados():New(nTop, nLeft, nBottom, nRight, nOpc, cLinhaOk, cTudoOk, cIniCpos, lDelete, aAlter, uPar1, lEmpty, nMax, cFieldOk, cSuperDel, uPar2, cDelOk, oWnd)

☑ **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	<p>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:</p> <p>{cTitulo, cRotina, nOpção, nAcesso}, aonde:</p> <p>nOpção segue o padrão do ERP Protheus para:</p> <ul style="list-style-type: none"> 6- Pesquisar 7- Visualizar 8- Incluir 9- Alterar 10- Excluir
aHeader	<p>Vetor com informações das colunas no formato:</p> <p>{cTitulo, cCampo, cPicture, nTamanho, nDecimais,; cValidação, cReservado, cTipo, xReservado1, xReservado2}</p> <p>A tabela temporária utilizada pela MsGetDB deverá ser criada com base no aHeader mais um último campo tipo lógico que determina se a linha foi excluída.</p>
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()	Oculto 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.

```

13.2.3. 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:**

Pedidos de Venda - Incluir

Cliente Loja
 Cli.Entrega Loja Entrega
 Transp. Tipo Cliente
 Cond. Pagto Tabela
 Vendedor 1 Comissao 1 0,00
 Vendedor 2 Comissao 2 0,00

Item	Produto	Unidade	Quantidade	Prc Unitario	Vir.Total	Qtd.Lib
			0,00	0,00	0,00	

☑ **Variáveis private:**

aRotina	<p>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:</p> <p>{cTitulo, cRotina, nOpção, nAcesso}, aonde:</p> <p>nOpção segue o padrão do ERP Protheus para:</p> <ol style="list-style-type: none"> 1- Pesquisar 2- Visualizar 3- Incluir 4- Alterar 5- Excluir
aHeader	<p>Vetor com informações das colunas no formato:</p> <p>{cTitulo, cCampo, cPicture, nTamanho, nDecimais,; cValidação, cReservado, cTipo, xReservado1, xReservado2}</p> <p>A tabela temporária utilizada pela MsGetDB deverá ser criada com base no aHeader mais um último campo tipo lógico que determina se a linha foi excluída.</p>
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()	Oculto 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        := "AlwaysTrue"
Local cTudoOk       := "AlwaysTrue"
Local cIniCpos      := "C6_ITEM"
Local nFreeze       := 000
Local nMax          := 999
Local cFieldOk      := "AlwaysTrue"
Local cSuperDel     := ""
Local cDelOk        := "AlwaysFalse"
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,"AlwaysTrue()",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
```

13.2.3.1. 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:

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

☑ Métodos adicionais:

SetBkBackColor	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:

A imagem mostra a interface de usuário 'Pedidos de Venda - Incluir'. No topo, há uma barra de título azul com o ícone de uma pasta e o texto 'Pedidos de Venda - Incluir'. Abaixo da barra, há uma barra de ferramentas com ícones de arquivo, pasta, lupa, seta verde, seta vermelha e seta amarela. O formulário contém campos para 'Cliente', 'Loja', 'Cli. Entrega', 'Loja Entrega', 'Transp.', 'Tipo Cliente', 'Cond. Pagto', 'Tabela', 'Vendedor 1', 'Comissao 1', 'Vendedor 2', 'Comissao 2'. Abaixo dos campos, há uma tabela com 7 colunas: 'Item', 'Produto', 'Unidade', 'Quantidade', 'Prc Unitario', 'Vlr. Total' e 'Qtd. Lib.'. A tabela contém duas linhas de dados com produtos 900001 e 900002. A interface é fechada por uma barra azul na base.

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 := "AlwaysTrue"
Local cTudoOk := "AlwaysTrue"
Local cIniCpos := "C6_ITEM"
Local nFreeze := 000
Local nMax := 999
```

Exemplo (continuação):

```
Local cFieldOk           := "AlwaysTrue"
Local cSuperDel          := ""
Local cDelOk             := "AlwaysFalse"
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,"AlwaysTrue()",;
                    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

13.3. 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 instânciação 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 instânciação direta de um objeto barra de botões genérico, o qual pode ser utilizado em qualquer posição da tela, tornando-o disponível na aplicação chamadora.

13.3.1. 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, IMsgDel, 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
IMsgDel	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

13.3.2. 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.

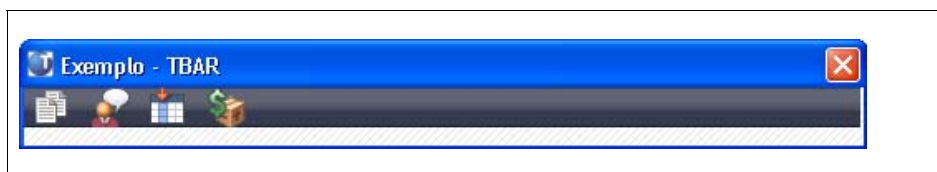
☑ **Sintaxe:** New(oWnd, nBtnWidth, nBtnHeight, l3D, cMode, oCursor, cResource, lNoAutoAdjust)

☑ **Retorno:** oTBar → objeto do tipo TBar()

☑ **Parâmetros:**

oWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
nBtnWidth	Númérico, opcional. Largura do botão contido na barra
nBtnHeight	Númérico, 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:lcCentered := .T.
oDlg:Activate()

Return

```

13.3.3. 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**

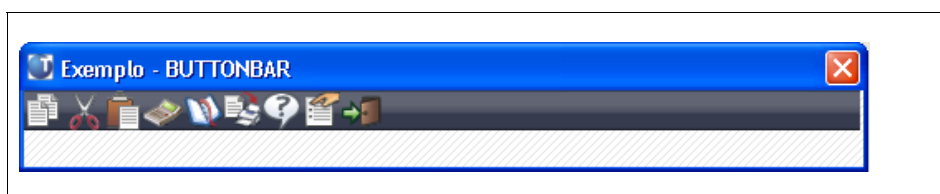
☒ **Sintaxe adicional:**

```
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 := {|| AlwaysTrue()}
oDlg:lCentered := .T.
oDlg:Activate()

Return
```



Anotações

13.3.4. 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
	EDIT
	EXCLUIR
	GRAF2D
	LINE
	OBJETIVO
	PENDENTE
	PRODUTO
	S4WB001N
	S4WB006N
	S4WB008N
	S4WB010N
	S4WB013N
	S4WB016N
	VENDEDOR

	DISCAGEM
	EDITABLE
	FORM
	GRAF3D
	NOTE
	OK
	PRECO
	S4SB014N
	S4WB005N
	S4WB007N
	S4WB009N
	S4WB011N
	S4WB014A
	SIMULACA
	USER

☑ **Exemplo:**

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

APÊNDICES

BOAS PRÁTICAS DE PROGRAMAÇÃO

14. 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
```

15. Utilização de Identação

É obrigatória a utilização da indentação, 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 indentação seguindo as estruturas de controle de fluxo (while, IF, caso etc.) torna a compreensão do código muito mais fácil:

```
While !SB1->(Eof())

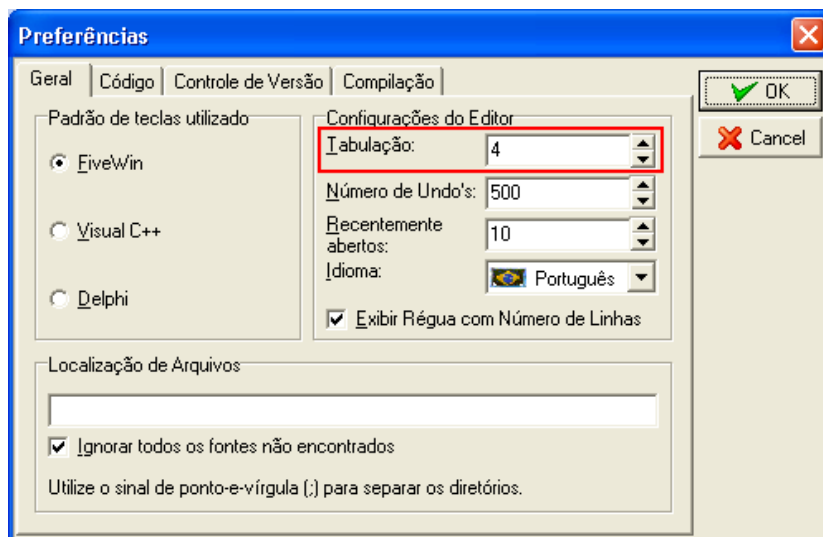
    If mv_par01 == SB1->B1_COD
        dbSkip()
        Loop
    Endif

    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 indentar o código utilize a tecla <TAB> e na ferramenta DEV-Studio, a qual pode ser configurada através da opção "Preferências":



16. 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()**

16.1. 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...
```

17. 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

18. 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, poucas 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 requirite 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 indentação 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 indentação

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-indentação de código. Para maiores detalhes, consulte a documentação sobre a indentação de códigos fontes disponível nos demais tópicos deste material.

Utilização de espaços em branco

Espaços em branco extras tornam o código mais fácil para a leitura. Não é necessário imensas áreas em branco, mas agrupar pedaços de código através da utilização de espaços em branco funciona muito bem. Costuma-se separar parâmetros com espaços em branco.

Quebra de linhas muito longas

Com o objetivo de tornar o código mais fácil de ler e imprimir, as linhas do código não devem estender o limite da tela ou do papel. Podem ser "quebradas" em mais de uma linha de texto utilizando o ponto-e-vírgula (;).

Capitulação de palavras-chave

Uma convenção amplamente utilizada é a de capitular as palavras chaves, funções, variáveis e campos utilizando uma combinação de caracteres em maiúsculo e minúsculo, visando facilitar a leitura do código fonte.

Avaliando o código a seguir:

```
local ncnt
while ( ncnt++ < 10 )
ntotal += ncnt * 2
enddo
```

O mesmo ficaria muito mais claro se re-escrito conforme abaixo:

```
Local nCnt  
While ( nCnt++ < 10 )  
    nTotal += nCnt * 2  
EndDo
```

Utilização da Notação Húngara

A Notação Húngara é muito comum entre programadores xBase e de outras linguagens. A documentação do ADVPL utiliza esta notação para a descrição das funções e comandos e é aconselhável sua utilização na criação de rotinas, pois ajuda a evitar pequenos erros e facilita a leitura do código. Para maiores detalhes, consulte a documentação sobre a utilização da Notação Húngara de códigos fontes disponível nos demais tópicos deste material.

Utilização de nomes significantes para variáveis

A principal vantagem da liberdade na criação dos nomes de variáveis é a facilidade de identificação da sua utilidade. Portanto deve-se utilizar essa facilidade o máximo possível. Nomes sem sentido apenas tornarão difícil a identificação da utilidade de uma determinada variável, assim como nomes extremamente curtos. Nem sempre a utilização de uma variável chamada *i* é a melhor saída. Claro, não convém criar uma variável com um nome muito longo que será utilizada como um contador, e referenciada muitas vezes no código. O bom senso deve ser utilizado.

Criar variáveis como *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.



Educação Corporativa

Guia de Referência Rápida ADVPL Intermediário



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

GUIA DE REFERÊNCIA RÁPIDA: Funções e Comandos ADVPL

Neste guia de referência rápida serão descritas as funções básicas da linguagem ADVPL, incluindo as funções herdadas da linguagem Clipper, necessárias ao desenvolvimento no ambiente ERP.

Conversão entre tipos de dados

CTOD()

Realiza a conversão de uma informação do tipo caracter no formato "DD/MM/AAAA" para uma variável do tipo data.

- ☑ **Sintaxe: CTOD(cData)**
- ☑ **Parâmetros**

cData	Caracter no formato "DD/MM/AAAA"
--------------	----------------------------------

Exemplo:

```
cData := "31/12/2006"
dData := CTOD(cData)

IF dDataBase >= dData
    MSGALERT("Data do sistema fora da competência")
ELSE
    MSGINFO("Data do sistema dentro da competência")
ENDIF
```

CVALTOCHAR()

Realiza a conversão de uma informação do tipo numérico em uma string, sem a adição de espaços a informação.

- ☑ **Sintaxe: CVALTOCHAR(nValor)**
- ☑ **Parâmetros**

nValor	Valor numérico que será convertido para caractere.
---------------	--

Exemplo:

```
FOR nPercorridos := 1 to 10
    MSGINFO("Passos percorridos: "+CvalToChar(nPercorridos))
NEXT nPercorridos
```

DTOC()

Realiza a conversão de uma informação do tipo data para em caracter, sendo o resultado no formato "DD/MM/AAAA".

- ☒ **Sintaxe: DTOC(dData)**
- ☒ **Parâmetros**

dData	Variável com conteúdo data
--------------	----------------------------

Exemplo:

```
MSGINFO("Database do sistema: "+DTOC(dData)
```

DTOS()

Realiza a conversão de uma informação do tipo data em um caracter, sendo o resultado no formato "AAAAMMDD".

- ☒ **Sintaxe: DTOS(dData)**
- ☒ **Parâmetros**

dData	Variável com conteúdo data
--------------	----------------------------

Exemplo:

```
cQuery := "SELECT A1_COD, A1_LOJA, A1_NREDUZ FROM SA1010 WHERE "  
cQuery += "A1_DULTCOM >="+"DTOS(dDataIni)+"
```

STOD()

Realiza a conversão de uma informação do tipo caracter com conteúdo no formato "AAAAMMDD" em data.

- ☒ **Sintaxe: STOD(sData)**
- ☒ **Parâmetros**

sData	String no formato "AAAAMMDD"
--------------	------------------------------

Exemplo:

```
sData := LERSTR(01,08) // Função que realiza a leitura de uma string de um txt previamente  
// aberto  
dData := STOD(sData)
```

STR()

Realiza a conversão de uma informação do tipo numérico em uma string, adicionando espaços à direita.

- ☑ **Sintaxe: STR(nValor)**
- ☑ **Parâmetros**

nValor	Valor numérico que será convertido para caractere.
---------------	--

Exemplo:

```
FOR nPercorridos := 1 to 10
    MSGINFO("Passos percorridos: "+CvalToChar(nPercorridos))
NEXT nPercorridos
```

STRZERO()

Realiza a conversão de uma informação do tipo numérico em uma string, adicionando zeros à esquerda do número convertido, de forma que a string gerada tenha o tamanho especificado no parâmetro.

- ☑ **Sintaxe: STRZERO(nValor, nTamanho)**
- ☑ **Parâmetros**

nValor	Valor numérico que será convertido para caractere.
nTamanho	Tamanho total desejado para a string retornada.

Exemplo:

```
FOR nPercorridos := 1 to 10
    MSGINFO("Passos percorridos: "+CvalToChar(nPercorridos))
NEXT nPercorridos
```

VAL()

Realiza a conversão de uma informação do tipo caracter em numérica.

- ☒ **Sintaxe:** VAL(cValor)
- ☒ **Parâmetros**

cValor	String que será convertida para numérico.
---------------	---

Exemplo:

```
Static Function Modulo11(cData)
LOCAL L, D, P := 0
L := Len(cdata)
D := 0
P := 1
While L > 0
    P := P + 1
    D := D + (Val(SubStr(cData, L, 1)) * P)
    If P = 9
        P := 1
    End
    L := L - 1
End
D := 11 - (mod(D,11))
If (D == 0 .Or. D == 1 .Or. D == 10 .Or. D == 11)
    D := 1
End
Return(D)
```



Anotações

Matemáticas

ACOS()

Função utilizada para calcular o valor do arco co-seno.

- ☒ **Sintaxe: ACOS(nValor)**
- ☒ **Parâmetros:**

nValor	Valor entre -1 e 1 de quem será calculado o Arco Co-Seno.
---------------	---

- ☒ **Retorno:**

Numérico	Range de 0 a π radianos. Se o valor informado no parâmetro for menor que -1 ou maior que 1, acos retorna um valor indefinido por default $[+\infty, -\infty]$
-----------------	---

CEILING()

Função utilizada para calcular o valor mais próximo possível de um valor nMax informado como parâmetro para a função.

- ☒ **Sintaxe: CEILING(nMax)**
- ☒ **Parâmetros:**

nMax	Valor limite para análise da função, no formato floating-point.
-------------	---

- ☒ **Retorno:**

Numérico	Valor do tipo double, representando o menor inteiro que é maior ou igual ao valor de nX. Não há retorno de erro na função.
-----------------	--

COS()

Função utilizada para calcular o valor do co-seno ou co-seno hiperbólico.

Importante: Se $x \geq 2^{63}$ ou $x \leq -2^{63}$ ocorre perda significativa na chamada da função COS().

- ☒ **Sintaxe: COS(nAngulo)**
- ☒ **Parâmetros:**

nAngulo	Valor que representa o ângulo em radianos.
----------------	--

- ☒ **Retorno:**

Numérico	Valor que representa o co-seno ou co-seno hiperbólico do ângulo informado.
-----------------	--

☒ **Situações inválidas:**

Entrada	Exceção apresentada	Significado da Exceção
\pm QNAN,IND	None	Sem Domínio
$\pm \infty$ (cosf, cos)	INVALID	Sem Domínio
$x \geq 7.104760e+002$ (cosh, coshf)	INEXACT+OVERFLOW	OVERFLOW

LOG10()

Função utilizada para calcular o logaritmo natural de um valor numérico, em base 10.

LOG10() é uma função numérica que calcula o logaritmo natural de um número. O logaritmo natural tem como base o valor 10. Devido ao arredondamento matemático, os valores retornados por LOG() podem não coincidir exatamente.

☒ **Sintaxe: LOG10(nNatural)**

☒ **Parâmetros:**

nNatural	Valor cujo o logaritmo deve ser encontrado.
-----------------	---

☒ **Retorno:**

Numérico	A função retorna o logaritmo de nNatural se bem sucedidas. Se nNatural for negativo, estas funções retornam um indefinido, pelo defeito. Se nNatural for 0, retornam INF(infinito).
-----------------	---

SIN()

Função utilizada para calcular o valor do seno ou seno hiperbólico. Devemos informar como parâmetro para a função um valor que representa o ângulo em radianos.

Importante: Se $x \geq 2^{63}$ ou $x \leq -2^{63}$ ocorre perda significativa na chamada da função SIN().

☒ **Sintaxe: SIN(nAngulo)**

☒ **Parâmetros:**

nAngulo	Valor do ângulo em radianos.
----------------	------------------------------

☒ **Retorno:**

Numérico	Retorna o valor do seno do ângulo especificado.
-----------------	---

☒ **Situações inválidas:**

Entrada	Exceção apresentada	Significado da Exceção
\pm QNAN,IND	None	Sem Domínio
$\pm \infty$ (senf, sen)	INVALID	Sem Domínio
$x \geq 7.104760e+002$ (senh, senhf)	INEXACT+OVERFLOW	OVERFLOW

SQRT()

Função utilizada para calcular a raiz quadrada de um número positivo.

- ☒ **Sintaxe: SQRT(nValor)**
- ☒ **Parâmetros:**

nValor	Um número positivo do qual será calculada a raiz quadrada.
---------------	--

- ☒ **Retorno:**

Númérico	Retorna um valor numérico calculado com precisão dupla. A quantidade de casas decimais exibidas é determinada apenas por SET DECIMALS, sem importar a configuração de SET FIXED. Um número negativo <nValor> retorna zero.
-----------------	--

TAN()

Função utilizada para calcular o valor da tangente ou tangente hiperbólica.

Importante: Se $x \geq 2^{63}$ ou $x \leq -2^{63}$ ocorre perda significativa na chamada da função cos.

- ☒ **Sintaxe: TAN(nAngulo)**
- ☒ **Parâmetros:**

nAngulo	Valor do ângulo em radianos.
----------------	------------------------------

- ☒ **Retorno:**

Númérico	Retorna o valor da tangente do ângulo especificado.
-----------------	---

- ☒ **Situações inválidas:**

Entrada	Exceção apresentada	Significado da Exceção
\pm QNAN,IND	None	Sem Domínio
$\pm \infty$	INVALID	Sem Domínio



Anotações

Análise de variáveis

TYPE()

Determina o tipo do conteúdo de uma variável, a qual não foi definida na função em execução.

- ☒ **Sintaxe:** TYPE("cVariavel")
- ☒ **Parâmetros**

"cVariavel"	Nome da variável que se deseja avaliar, entre aspas ("").
--------------------	---

Exemplo:

```
IF TYPE("dDataBase") == "D"
    MSGINFO("Database do sistema: "+DTC("dDataBase"))
ELSE
    MSGINFO("Variável indefinida no momento")
ENDIF
```

VALTYPE()

Determina o tipo do conteúdo de uma variável, a qual não foi definida na função em execução.

- ☒ **Sintaxe:** VALTYPE(cVariavel)
- ☒ **Parâmetros**

cVariavel	Nome da variável que se deseja avaliar.
------------------	---

Exemplo:

```
STATIC FUNCTION GETTEXTO(nTamanho, cTitulo, cSay)
LOCAL cTexto      := ""
LOCAL nColF, nLargGet  := 0
PRIVATE oDlg
Default cTitulo    := "Tela para informar texto"
Default cSay       := "Informe o texto:"
Default nTamanho   := 1

nTamanho := IIF(ValType(nTamanho) != "N", 1, nTamanho) // Se o parâmetro foi passado
cTexto   := Space(nTamanho); nLargGet := Round(nTamanho * 2.5, 0);
nColF    := Round(195 + (nLargGet * 1.75), 0)

DEFINE MSDIALOG oDlg TITLE cTitulo FROM 000,000 TO 120,nColF PIXEL
@ 005,005 TO 060, Round(nColF/2,0) OF oDlg PIXEL
@ 010,010 SAY cSay SIZE 55, 7 OF oDlg PIXEL
@ 010,065 MSGET cTexto SIZE nLargGet, 11 OF oDlg PIXEL ;
Picture "@!" VALID !Empty(cTexto)
DEFINE SBUTTON FROM 030, 010 TYPE 1 ;
ACTION (nOpca := 1,oDlg:End()) ENABLE OF oDlg
DEFINE SBUTTON FROM 030, 040 TYPE 2 ;
ACTION (nOpca := 0,oDlg:End()) ENABLE OF oDlg
ACTIVATE MSDIALOG oDlg CENTERED
cTexto := IIF(nOpca==1,cTexto,"")
RETURN cTexto
```

Manipulação de arrays

AADD()

A função AADD() permite a inserção de um item em um array já existente, sendo que este item podem ser um elemento simples, um objeto ou outro array.

- ☑ **Sintaxe: AADD(aArray, xItem)**
- ☑ **Parâmetros**

aArray	Array pré-existente no qual será adicionado o item definido em xItem
xItem	Item que será adicionado ao array.

Exemplo:

```
aDados := {} // Define que a variável aDados é um array, sem especificar suas dimensões.
aItem := {} // Define que a variável aItem é um array, sem especificar suas dimensões.

AADD(aItem, cVariavel1) // Adiciona um elemento no array aItem de acordo com o cVariavel1
AADD(aItem, cVariavel2) // Adiciona um elemento no array aItem de acordo com o cVariavel2
AADD(aItem, cVariavel3) // Adiciona um elemento no array aItem de acordo com o cVariavel3

// Neste ponto o array aItem possui 03 elementos os quais podem ser acessados com:
// aItem[1] -> corresponde ao conteúdo de cVariavel1
// aItem[2] -> corresponde ao conteúdo de cVariavel2
// aItem[3] -> corresponde ao conteúdo de cVariavel3

AADD(aDados,aItem) // Adiciona no array aDados o conteúdo do array aItem

// Neste ponto, o array aDados possui apenas um elemento, que também é um array
// contendo 03 elementos:
// aDados [1][1] -> corresponde ao conteúdo de cVariavel1
// aDados [1][2] -> corresponde ao conteúdo de cVariavel2
// aDados [1][3] -> corresponde ao conteúdo de cVariavel3

AADD(aDados, aItem)
AADD(aDados, aItem)

// Neste ponto, o array aDados possui 03 elementos, aonde cada qual é um array com outros
// 03 elementos, sendo:

// aDados [1][1] -> corresponde ao conteúdo de cVariavel1
// aDados [1][2] -> corresponde ao conteúdo de cVariavel2
// aDados [1][3] -> corresponde ao conteúdo de cVariavel3

// aDados [2][1] -> corresponde ao conteúdo de cVariavel1
// aDados [2][2] -> corresponde ao conteúdo de cVariavel2
// aDados [2][3] -> corresponde ao conteúdo de cVariavel3

// aDados [3][1] -> corresponde ao conteúdo de cVariavel1
// aDados [3][2] -> corresponde ao conteúdo de cVariavel2
// aDados [3][3] -> corresponde ao conteúdo de cVariavel3
```

Exemplo (continuação):

```
// Desta forma, o array aDados montando com uma estrutura de 03 linhas e 03 colunas, com  
// o conteúdo definido por variáveis externas, mas com a mesma forma obtida com o uso do  
// comando: aDados := ARRAY(3,3).
```

ACLONE()

A função ACLONE() realiza a cópia dos elementos de um array para outro array integralmente.

- ☑ **Sintaxe: AADD(aArray)**
- ☑ **Parâmetros**

aArray	Array pré-existente que terá seu conteúdo copiado para o array especificado.
---------------	--

Exemplo:

```
// Utilizando o array aDados utilizado no exemplo da função AADD()  
aItens := ACLONE(aDados)  
  
// Neste ponto, o array aItens possui exatamente a mesma estrutura e informações do array  
// aDados.
```



Importante

Por ser uma estrutura de memória, um array não pode ser simplesmente copiado para outro array através de uma atribuição simples (":=").

Para mais informações sobre a necessidade de utilizar o comando ACLONE() verifique o tópico 6.1.3 – Cópia de Arrays.

ACOPY()

Função de array que copia elementos do array aOrigem para array aDestino. O array destino aDestino já deve ter sido declarado e grande o bastante para conter os elementos que serão copiados. Se o array aOrigem contiver mais elementos, alguns dos elementos não serão copiados. ACOPY() copia os valores de todos os dados, incluindo valores nulos (NIL) e códigos de bloco.

Se um elemento for um subarray, o elemento correspondente no array aDestino, conterá o mesmo subarray. Portanto, ACOPY() não produzirá uma cópia completa de array multidimensionais.

- ☑ **Sintaxe: ACOPY(aOrigem, aDestino , [nInicio], [nQtde], [nPosDestino])**
- ☑ **Parâmetros:**

aOrigem	é o array que contém os elementos a serem copiados.
aDestino	é o array que receberá a cópia dos elementos.
nInicio	indica qual o índice do primeiro elemento de <i>aOrigem</i> que será copiado. Se não for especificado, o valor assumido será 01.

nQtde	indica a quantidade de elementos a serem copiados a partir do array <i>aOrigem</i> . iniciando-se a contagem a partir da posição <i>nInicio</i> . Se <i>nQtde</i> não for especificado, todos os elementos do array <i>aOrigem</i> serão copiados, iniciando-se a partir da posição <i>nInicio</i> .
nPosDestino	é a posição do elemento inicial no array <i>aDestino</i> que receberá os elementos de <i>aOrigem</i> . Se não especificado, será assumido 01.

☒ **Retorno:**

aDestino	referência ao array <i>aDestino</i> .
-----------------	---------------------------------------

Exemplo:

```
LOCAL nCount := 2, nStart := 1, aOne, aTwo
aOne := { 1, 1, 1 }
aTwo := { 2, 2, 2 }
ACOPY(aOne, aTwo, nStart, nCount)
// Result: aTwo is now { 1, 1, 2 }
```

ADEL()

A função ADEL() permite a exclusão de um elemento do array. Ao efetuar a exclusão de um elemento, todos os demais são reorganizados de forma que a última posição do array passará a ser nula.

☒ **Sintaxe: ADEL(aArray, nPosição)**

☒ **Parâmetros**

aArray	Array do qual deseja-se remover uma determinada posição
nPosição	Posição do array que será removida

Exemplo:

// Utilizando o array aItens do exemplo da função ACLONE() temos:

ADEL(aItens,1) // Será removido o primeiro elemento do array aItens.

```
// Neste ponto, o array aItens continua com 03 elementos, aonde:
// aItens[1] -> antigo aItens[2], o qual foi reordenado como efeito da exclusão do item 1.
// aItens[2] -> antigo aItens[3], o qual foi reordenado como efeito da exclusão do item 1.
// aItens[3] -> conteúdo nulo, por se tratar do item excluído.
```

ADIR()

Função que preenche os arrays passados com os dados dos arquivos encontrados, através da máscara informada. Tanto arquivos locais (Remote) como do servidor podem ser informados.

Importante: *ADir é uma função obsoleta, utilize sempre Directory().*

☑ **Sintaxe:** ADIR([*cArqEspec*], [*aNomeArq*], [*aTamanho*], [*aData*], [*aHora*], [*aAtributo*])

☑ **Parâmetros:**

cArqEspec	Caminho dos arquivos a serem incluídos na busca de informações. Segue o padrão para especificação de arquivos, aceitando arquivos no servidor Protheus e no Cliente. Caracteres como * e ? são aceitos normalmente. Caso seja omitido, serão aceitos todos os arquivos do diretório default (*.*).
aNomeArq	Array de Caracteres. É o array com os nomes dos arquivos encontrados na busca.O conteúdo anterior do array é apagado.
aTamanho	Array Numérico. São os tamanhos dos arquivos encontrados na busca.
aData	Array de Datas. São as datas de modificação dos arquivos encontrados na busca.
aHora	Array de Caracteres. São os horários de modificação dos arquivos encontrados. Cada elemento contém horário no formato: hh:mm:ss.
aAtributos	Array de Caracteres. São os atributos dos arquivos, caso esse array seja passado como parâmetros, serão incluídos os arquivos com atributos de sistema e ocultos.

☑ **Retorno:**

nArquivos	Quantidade de arquivos encontrados.
------------------	-------------------------------------

Exemplo:

```
LOCAL aFiles[ADIR("*.TXT")]
ADIR("*.TXT", aFiles)
AEVAL(aFiles, { |element| QOUT(element) })
```



Anotações

AFILL()

Função de manipulação de arrays, que preenche os elementos do array com qualquer tipo de dado. Incluindo code-block. Esta função não deve ser usada para preencher um array com outro array.

- ☑ **Sintaxe:** **AFILL(aDestino , xExpValor, [nInicio], [nQuantidade])**
- ☑ **Parâmetros**

aDestino	É o onde os dados serão preenchidos.
xExpValor	É o dado que será preenchido em todas as posições informadas, não é permitida a utilização de arrays.
nInicio	É a posição inicial de onde os dados serão preenchidos, o valor padrão é 1.
nCount	Quantidade de elementos a partir de [nInicio] que serão preenchidos com <expValor>, caso não seja informado o valor será a quantidade de elementos até o final do array.

- ☑ **Retorno:**

aDestino	Retorna uma referência para aDestino.
-----------------	---------------------------------------

Exemplo:

```
LOCAL aLogic[3]
// Resultado: aLogic é { NIL, NIL, NIL }
AFILL(aLogic, .F.)
// Resultado: aLogic é { .F., .F., .F. }
AFILL(aLogic, .T., 2, 2)
// Resultado: aLogic é { .F., .T., .T. }
```

AINS()

A função AINS() permite a inserção de um elemento no array especificado em qualquer ponto da estrutura do mesmo, diferindo desta forma da função AADD() a qual sempre insere um novo elemento ao final da estrutura já existente.

- ☑ **Sintaxe:** **AINS(aArray, nPosicao)**
- ☑ **Parâmetros**

aArray	Array pré-existente no qual desejasse inserir um novo elemento.
nPosicao	Posição na qual o novo elemento será inserido.

Exemplo:

```
aAlunos := {"Edson", "Robson", "Renato", "Tatiana"}
AINS(aAlunos,3)
// Neste ponto o array aAlunos terá o seguinte conteúdo:
// {"Edson", "Robson", nulo, "Renato", "Tatiana"}
```



Similar ao efeito da função ADEL(), o elemento inserido no array pela função AINS() terá um conteúdo nulo, sendo necessário trata-lo após a realização deste comando.

ARRAY()

A função Array() é utilizada na definição de variáveis de tipo array, como uma opção a sintaxe utilizando chaves ("{}").

- ☑ **Sintaxe: Array(nLinhas, nColunas)**
- ☑ **Parâmetros**

nLinhas	Determina o número de linhas com as quais o array será criado
nColunas	Determina o número de colunas com as quais o array será criado

Exemplo:

```
aDados := Array(3,3) // Cria um array de três linhas, cada qual com 3 colunas.
```



O array definido pelo comando Array() apesar de já possuir a estrutura solicitada, não possui conteúdo em nenhum de seus elementos, ou seja:

```
aDados[1] -> array de três posições  
aDados[1][1] -> posição válida, mas de conteúdo nulo.
```

ASCAN()

A função ASCAN() permite que seja identificada a posição do array que contém uma determinada informação, através da análise de uma expressão descrita em um bloco de código.

- ☑ **Sintaxe: ASCAN(aArray, bSeek)**
- ☑ **Parâmetros**

aArray	Array pré-existente no qual desejasse identificar a posição que contém a informação pesquisada.
bSeek	Bloco de código que configura os parâmetros da busca a ser realizada.

Exemplo:

```
aAlunos := {"Márcio", "Denis", "Arnaldo", "Patrícia"}  
bSeek := {|x| x == "Denis"}  
nPosAluno := aScan(aAlunos,bSeek) // retorno esperado -> 2
```




Dica

Durante a execução da função aScan, a variável "x" receberá o conteúdo o item que está posicionado no momento, no caso aAlunos[x]. Como aAlunos[x] é uma posição do array que contém o nome do aluno, "x" poderia ser renomeada para cNome, e a definição do bloco bSeek poderia ser re-escrita como:

```
bSeek := { |cNome| cNome == "Denis" }
```



Dica

Na definição dos programas é sempre recomendável utilizar variáveis com nomes significativos, desta forma os blocos de código não são exceção.

Sempre opte por analisar como o bloco de código será utilizado e ao invés de "x", "y" e similares, defina os parâmetros com nomes que representem seu conteúdo. Será mais simples o seu entendimento e o entendimento de outros que forem analisar o código escrito.

ASCANX()

Função utilizada para varrer um vetor procurando um valor especificado, operando de forma similar a função ASCAN.

A diferença fundamental da função ASCANX é que esta função recebe um segundo parâmetro em seu code-block representando o índice do array.

☑ **Sintaxe:** ASCANX (< xDestino > , < bSeek > , [nInicio] , [nCont])

☑ **Parâmetros:**

xDestino	Representa o objeto a ser varrido pela função, pode ser atribuído ao parâmetro um array um Objeto.
bSeek	Representa o valor que será pesquisado, podendo ser um bloco de código.
nInicio	Representa o elemento a partir do qual terá início a pesquisa, quando este argumento não for informado o valor default será 1.
nCont	Representa a quantidade de elementos que serão pesquisados a partir da posição inicial, quando este argumento não for informado todos elementos do array serão pesquisados.

Exemplo.:

```
nPos := aScanX( ARRAY, { |X,Y| X[1] == cNome .OR. y<=100} )
```



Dica

No código demonstrado acima, note a inclusão no code-block do Y, onde a função irá terminar sua execução em 3 condições:

- 1) Até encontrar o elemento no ARRAY com a ocorrência cNome, retornando a posição desse elemento.
- 2) Essa é novidade, ASCANX irá verificar o Array até a posição 100.
- 3) O elemento cNome não foi encontrado no ARRAY e a condição de Y até 100 não satisfaz, pois o array é menor do que 100 posições!



Dica

Como ASCAN() que utiliza o operador (=) para comparações, a função ASCANX() também é case sensitive, no caso os elementos procurados devem ser exatamente igual.

ASIZE()

A função ASIZE permite a redefinição da estrutura de um array pré-existente, adicionando ou removendo itens do mesmo.

☑ **Sintaxe: ASIZE(aArray, nTamanho)**

☑ **Parâmetros**

aArray	Array pré-existente que terá sua estrutura redimensionada.
nTamanho	Tamanho com o qual deseja-se redefinir o array. Se o tamanho for menor do que o atual, serão removidos os elementos do final do array, já se o tamanho for maior do que o atual serão inseridos itens nulos ao final do array.

Exemplo:

```
// Utilizando o array aItens, o qual teve um elemento excluído pelo uso da função ADEL()
```

```
ASIZE(aItens,Len(aItens-1))
```

```
// Neste ponto o array aItens possui 02 elementos, ambos com conteúdos válidos.
```



Dica

Utilizar a função ASIZE() após o uso da função ADEL() é uma prática recomendada e evita que seja acessada uma posição do array com um conteúdo inválido para a aplicação em uso.

ASORT()

A função ASORT() permite que os itens de um array sejam ordenados a partir de um critério pré-estabelecido.

- ☑ **Sintaxe: ASORT(aArray, nInicio, nItens, bOrdem)**
- ☑ **Parâmetros**

aArray	Array pré-existente que terá seu conteúdo ordenado através de um critério estabelecido.
nInicio	Posição inicial do array para início da ordenação. Caso não seja informado, o array será ordenado a partir de seu primeiro elemento.
nItens	Quantos itens, a partir da posição inicial deverão ser ordenados. Caso não seja informado, serão ordenados todos os elementos do array.
bOrdem	Bloco de código que permite a definição do critério de ordenação do array. Caso bOrdem não seja informado, será utilizado o critério ascendente.



Dica

Um bloco de código é basicamente uma função escrita em linha. Desta forma sua estrutura deve "suportar" todos os requisitos de uma função, os quais são através da análise e interpretação de parâmetros recebidos, executar um processamento e fornecer um retorno.

Com base nesse requisito, pode-se definir um bloco de código com a estrutura abaixo:

bBloco := { |xPar1, xPar2, ... xParZ| Ação1, Ação2, AçãoZ } , aonde:

|| -> define o intervalo onde estão compreendidos os parâmetros

Ação Z-> expressão que será executadas pelo bloco de código

Ação1... AçãoZ -> intervalo de expressões que serão executadas pelo bloco de código, no formato de lista de expressões.

Retorno -> resultado da ultima ação executada pelo bloco de código, no caso AçãoZ.

Para maiores detalhes sobre a estrutura e utilização de blocos de código consulte o tópico 6.2 – Listas de Expressões e Blocos de código.

Exemplo 01 – Ordenação ascendente

```
aAlunos := { "Mauren", "Soraia", "Andréia" }
```

```
aSort(aAlunos)
```

```
// Neste ponto, os elementos do array aAlunos serão {"Andréia", "Mauren", "Soraia" }
```

Exemplo 02 – Ordenação descendente

```
aAlunos := { "Mauren", "Soraia", "Andréia"}  
bOrdem := { |x,y| x > y }  
  
// Durante a execução da função aSort(), a variável "x" receberá o conteúdo do item que está  
// posicionado. Como o item que está posicionado é a posição aAlunos[x] e aAlunos[x] ->  
// string contendo o nome de um aluno, pode-se substituir "x" por cNomeAtu.  
// A variável "y" receberá o conteúdo do próximo item a ser avaliado, e usando a mesma  
// analogia de "x", pode-se substituir "y" por cNomeProx. Desta forma o bloco de código  
// bOrdem pode ser re-escrito como:  
  
bOrdem := { |cNomeAtu, cNomeProx| cNomeAtu > cNomeProx }  
  
aSort(aAlunos, bOrdem)  
  
// Neste ponto, os elementos do array aAlunos serão {"Soraia", "Mauren", "Andréia"}
```

ATAILO

ATAIL() é uma função de manipulação de array que retorna o último elemento de um array. Ela deve ser usada em substituição da seguinte construção: aArray [LEN(aArray)]

☒ **Sintaxe: ATAIL(aArray)**

☒ **Parâmetros:**

aArray	É o array de onde será retornado o último elemento.
---------------	---

☒ **Retorno:**

nUltimo	Número do último elemento do array.
----------------	-------------------------------------

Exemplo:

```
aArray := {"a", "b", "c", "d"}  
ATAIL(aArray) // Resultado: d
```



Anotações

Manipulação de blocos de código

EVAL()

A função EVAL() é utilizada para avaliação direta de um bloco de código, utilizando as informações disponíveis no mesmo de sua execução. Esta função permite a definição e passagem de diversos parâmetros que serão considerados na interpretação do bloco de código.

- ☒ **Sintaxe:** EVAL(bBloco, xParam1, xParam2, xParamZ)
- ☒ **Parâmetros**

bbloco	Bloco de código que será interpretado.
xParamZ	Parâmetros que serão passados ao bloco de código. A partir da passagem do bloco, todos os demais parâmetros da função serão convertidos em parâmetros para a interpretação do código.

Exemplo:

```
nInt := 10
bBloco := {|N| x:= 10, y:= x*N, z:= y/(x*N)}
nValor := EVAL(bBloco, nInt)
// O retorno será dado pela avaliação da ultima ação da lista de expressões, no caso "z".
// Cada uma das variáveis definidas em uma das ações da lista de expressões fica disponível
// para a próxima ação.
// Desta forma temos:
// N → recebe nInt como parâmetro (10)
// X → tem atribuído o valor 10 (10)
// Y → resultado da multiplicação de X por N (100)
// Z → resultado a divisão de Y pela multiplicação de X por N ( 100 / 100) → 1
```

DBEVAL()

A função DBEval() permite que todos os registros de uma determinada tabela sejam analisados e para cada registro será executado o bloco de código definido.

- ☒ **Sintaxe:** Array(bBloco, bFor, bWhile)
- ☒ **Parâmetros**

bbloco	Bloco de código principal, contendo as expressões que serão avaliadas para cada registro do alias ativo.
bFor	Condição para continuação da análise dos registros, com o efeito de uma estrutura For ... Next.
bWhile	Condição para continuação da análise dos registros, com o efeito de uma estrutura While ... End

Exemplo 01:

// Considerando o trecho de código abaixo:

```
dbSelectArea("SX5")
dbSetOrder(1)
dbGotop()
```

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

// O mesmo pode ser re-escrito com o uso da função DBEVAL():

```
dbEval( { |x| nCnt++ },,{ ||X5_FILIAL==xFilial("SX5") .And. X5_TABELA<=mv_par02})
```

Exemplo 02:

// Considerando o trecho de código abaixo:

```
dbSelectArea("SX5")
dbSetOrder(1)
dbGotop()
```

```
While !Eof() .And. X5_TABELA == cTabela
    AADD(aTabela,{X5_CHAVE,Capital(X5_DESCRI)})
    dbSkip()
End
```

// O mesmo pode ser re-escrito com o uso da função DBEVAL():

```
dbEval({ || AADD(aTabela,{X5_CHAVE,Capital(X5_DESCRI)}) },,{ || X5_TABELA==cTabela})
```



Importante

Na utilização da função DBEVAL() deve ser informado apenas um dos dois parâmetros: bFor ou bWhile.



Anotações

AEVAL()

A função AEVAL() permite que todos os elementos de um determinada array sejam analisados e para cada elemento será executado o bloco de código definido.

- ☑ **Sintaxe:** AEVAL(aArray, bBloco, nInicio, nFim)
- ☑ **Parâmetros**

aArray	Array que será avaliado na execução da função.
bBloco	Bloco de código principal, contendo as expressões que serão avaliadas para cada elemento do array informado.
nInicio	Elemento inicial do array, a partir do qual serão avaliados os blocos de código.
nFim	Elemento final do array, até o qual serão avaliados os blocos de código.

Exemplo 01:

Considerando o trecho de código abaixo:

```
AADD(aCampos,"A1_FILIAL")
AADD(aCampos,"A1_COD")
SX3->(dbSetOrder(2))
For nX:=1 To Len(aCampos)
  SX3->(dbSeek(aCampos[nX]))
  AADD(aTitulos,AllTrim(SX3->X3_TITULO))
Next nX
```

O mesmo pode ser re-escrito com o uso da função AEVAL():

```
aEval(aCampos,{|x| SX3->(dbSeek(x)),IIF(Found(), AADD(aTitulos,;
AllTrim(SX3->X3_TITULO)))})
```



Anotações

Manipulação de strings

ALLTRIM()

Retorna uma string sem os espaços à direita e à esquerda, referente ao conteúdo informado como parâmetro.

A função ALLTRIM() implementa as ações das funções RTRIM ("right trim") e LTRIM ("left trim").

- ☒ **Sintaxe: ALLTRIM(cString)**
- ☒ **Parâmetros**

cString	String que será avaliada para remoção dos espaços a direita e a esquerda.
----------------	---

Exemplo:

```
cNome := ALLTRIM(SA1->A1_NOME)

MSGINFO("Dados do campo A1_NOME:" + CRLF
"Tamano:" + CVALTOCHAR(LEN(SA1->A1_NOME)) + CRLF
"Texto:" + CVALTOCHAR(LEN(cNome)))
```

ASC()

Converte uma informação caractere em seu valor de acordo com a tabela ASCII.

- ☒ **Sintaxe: ASC(cCaractere)**
- ☒ **Parâmetros**

cCaractere	Caracter que será consultado na tabela ASCII.
-------------------	---

Exemplo:

```
USER FUNCTION NoAcento(Arg1)
Local nConta := 0
Local cLetra := ""
Local cRet := ""
Arg1 := Upper(Arg1)
For nConta:= 1 To Len(Arg1)
    cLetra := SubStr(Arg1, nConta, 1)
    Do Case
Case (Asc(cLetra) > 191 .and. Asc(cLetra) < 198) .or.;
(Asc(cLetra) > 223 .and. Asc(cLetra) < 230)
    cLetra := "A"
Case (Asc(cLetra) > 199 .and. Asc(cLetra) < 204) .or.;
(Asc(cLetra) > 231 .and. Asc(cLetra) < 236)
    cLetra := "E"
```


Exemplo (continuação):

```
Case (Asc(cLetra) > 204 .and. Asc(cLetra) < 207) .or.;  
(Asc(cLetra) > 235 .and. Asc(cLetra) < 240)  
    cLetra := "I"  
  
Case (Asc(cLetra) > 209 .and. Asc(cLetra) < 215) .or.;  
(Asc(cLetra) == 240) .or. (Asc(cLetra) > 241 .and. Asc(cLetra) < 247)  
    cLetra := "O"  
  
Case (Asc(cLetra) > 216 .and. Asc(cLetra) < 221) .or.;  
(Asc(cLetra) > 248 .and. Asc(cLetra) < 253)  
    cLetra := "U"  
  
Case Asc(cLetra) == 199 .or. Asc(cLetra) == 231  
    cLetra := "C"  
  
EndCase  
  
    cRet := cRet+cLetra  
  
Next  
  
Return UPPER(cRet)
```

AT()

Retorna a primeira posição de um caractere ou string dentro de outra string especificada.

- ☒ **Sintaxe:** AT(cCaractere, cString)
- ☒ **Parâmetros**

cCaractere	Caractere ou string que se deseja verificar
cString	String na qual será verificada a existência do conteúdo de cCaractere.

Exemplo:

```
STATIC FUNCTION NOMASCARA(cString,cMascara,nTamanho)  
  
LOCAL cNoMascara := ""  
LOCAL nX := 0  
  
IF !Empty(cMascara) .AND. AT(cMascara,cString) > 0  
    FOR nX := 1 TO Len(cString)  
        IF !(SUBSTR(cString,nX,1) $ cMascara)  
            cNoMascara += SUBSTR(cString,nX,1)  
        ENDIF  
    NEXT nX  
    cNoMascara := PADR(ALLTRIM(cNoMascara),nTamanho)  
ELSE  
    cNoMascara := PADR(ALLTRIM(cString),nTamanho)  
ENDIF  
  
RETURN cNoMascara
```

BITON()

Função utilizada para ligar determinados bits de uma String passada por parâmetro para a função. Além da string à ser alterada, a função também recebe como parâmetro um número que indica o bit de início a ser alterado, um número que indica a quantidade de bits a serem alterados(ligados) e o tamanho da string passada.

- ☑ **Sintaxe:** BITON (< cValue > , < nBitIni > , < nBitEnd > , < nStrLen >)
- ☑ **Parâmetros**

cValue	String no qual desejamos ligar os bits.
nBitIni	Indica a partir de qual bit, começará a ser ligados os bits na String
nBitEnd	Indica a quantidade de bits que serão ligados a partir do início.
nStrLen	Representa o tamanho da String passada para a função.

CAPITAL()

Função que avalia a string passada como parâmetro alterando a primeira letra de cada palavra para maiúscula e as demais letras como minúsculas.

- ☑ **Sintaxe:** CAPITAL(cFrase)
- ☑ **Parâmetros:**

cFrase	String a ser avaliada
---------------	-----------------------

- ☑ **Retorno:**

String	Conteúdo da string original com as modificações necessárias para atender a condição da função.
---------------	--

CHR()

Converte um valor número referente a uma informação da tabela ASCII no caractere que esta informação representa.

- ☑ **Sintaxe:** CHR(nASCII)
- ☑ **Parâmetros**

nASCII	Código ASCII do caractere
---------------	---------------------------

Exemplo:

```
#DEFINE CRLF CHR(13)+CHR(10) // FINAL DE LINHA
```

DESCEND()

Função de conversão que retorna a forma complementada da expressão string especificada. Esta função normalmente é utilizada para a criação de indexadores em ordem decrescente

☑ **Sintaxe:** **DESCEND (< cString >)**

☑ **Parâmetros:**

cString	Corresponde à sequência de caracteres a ser analisada.
----------------	--

☑ **Retorno:**

Caracter	String complementada da string analisada.
-----------------	---

Exemplo:

```
// Este exemplo utiliza DESCEND() em uma expressão INDEX para criar um índice de datas de  
// ordem decrescente:
```

```
USE Sales NEW  
INDEX ON DESCEND(DTOS(OrdDate)) TO SalesDate
```

```
// Depois, DESCEND() pode ser utilizado para fazer uma pesquisa (SEEK) no índice  
// decrescente:
```

```
DbSEEK(DESCEND(DTOS(dFindDate)))
```

GETDTOVAL()

Função utilizada para retornar um numero formatado, de acordo com o valor passado por parâmetro, sendo que irá apenas manter os valores numéricos contidos na string passada por parâmetro, verificando se existe algum caractere '.' retornando um numero fracionário, na ordem dos números contidos na string.

A função é muito útil quando desejamos utilizar o valor numérico de uma data que está contida em uma string.

☑ **Sintaxe:** **GETDTOVAL (< cDtoVal >)**

☑ **Parâmetros:**

cDtoVal	Representa uma string contendo um valor numérico no qual será convertido.
----------------	---

☑ **Retorno:**

Numérico	Retorna um dado numérico de acordo com o valor informado em <cDtoVal>.
-----------------	--

Exemplo:

```
GetDtoVal('123456')      //retorno 123456.0000
GetDtoVal('1/2/3/4/5/6') //retorno 123456.0000
GetDtoVal('fim.123456')  //retorno 0.123456
GetDtoVal('teste')       //retorno 0.0
```

ISALPHA()

Função utilizada para determinar se o caractere mais à esquerda em uma cadeia de caracteres é alfabético, permitindo avaliar se o string especificado começa com um caractere alfabético. Um caractere alfabético consiste em qualquer letra maiúscula ou minúscula de "A" a "Z".

☒ **Sintaxe:** ISALPHA (< cString >)

☒ **Parâmetros:**

cString	Cadeia de caracteres a ser examinada.
----------------	---------------------------------------

☒ **Retorno:**

Lógico	Retorna verdadeiro (.T.) se o primeiro caractere em <cString> for alfabético, caso contrário, retorna falso (.F.).
---------------	--

ISDIGIT()

Função utilizada para determinar se o caractere mais à esquerda em uma cadeia de caracteres é um dígito, permitindo avaliar se o primeiro caractere em um string é um dígito numérico entre zero e nove.

☒ **Sintaxe:** ISDIGIT (< cString >)

☒ **Parâmetros:**

cString	Cadeia de caracteres a ser examinada.
----------------	---------------------------------------

☒ **Retorno:**

Lógico	Retorna verdadeiro (.T.) caso o primeiro caractere da cadeia seja um dígito entre zero e nove; caso contrário, retorna falso (.F.).
---------------	---

ISLOWER()

Função utilizada para determinar se o caractere mais à esquerda é uma letra minúscula, permitindo avaliar se o primeiro caractere de um string é uma letra minúscula. É o contrário de ISUPPER(), a qual determina se a cadeia de caracteres começa com uma letra maiúscula. ISLOWER() e ISUPPER() ambas são relacionadas às funções LOWER() e UPPER(), que convertem caracteres minúsculos para maiúsculos, e vice-versa.

☑ **Sintaxe:** ISLOWER(< cString >)

☑ **Parâmetros:**

cString	Cadeia de caracteres a ser examinada.
----------------	---------------------------------------

☑ **Retorno:**

Lógico	Retorna verdadeiro (.T.) caso o primeiro caractere da cadeia seja minúsculo , caso contrário, retorna falso (.F.).
---------------	--

ISUPPER()

Função utilizada para determinar se o caractere mais à esquerda é uma letra maiúscula, permitindo avaliar se o primeiro caractere de um string é uma letra maiúscula. É o contrário de ISLOWER(), a qual determina se a cadeia de caracteres começa com uma letra minúscula. ISLOWER() e ISUPPER() ambas são relacionadas às funções LOWER() e UPPER(), que convertem caracteres minúsculos para maiúsculos, e vice-versa.

☑ **Sintaxe:** ISUPPER(< cString >)

☑ **Parâmetros:**

cString	Cadeia de caracteres a ser examinada.
----------------	---------------------------------------

☑ **Retorno:**

Lógico	Retorna verdadeiro (.T.) caso o primeiro caractere da cadeia seja maiúsculo , caso contrário, retorna falso (.F.).
---------------	--

LEN()

Retorna o tamanho da string especificada no parâmetro.

☑ **Sintaxe:** LEN(cString)

☑ **Parâmetros**

cString	String que será avaliada
----------------	--------------------------

Exemplo:

```
cNome := ALLTRIM(SA1->A1_NOME)
MSGINFO("Dados do campo A1_NOME:" + CRLF
"Tamano:" + CVALTOCHAR(LEN(SA1->A1_NOME)) + CRLF
"Texto:" + CVALTOCHAR(LEN(cNome)))
```

LOWER()

Retorna uma string com todos os caracteres minúsculos, tendo como base a string passada como parâmetro.

- ☑ **Sintaxe: LOWER(cString)**
- ☑ **Parâmetros**

cString	String que será convertida para caracteres minúsculos.
----------------	--

Exemplo:

```
cTexto := "ADVPL"
```

```
MSGINFO("Texto:" + LOWER(cTexto))
```

LTRIM()

Função para tratamento de caracteres utilizada para formatar cadeias de caracteres que possuam espaços em branco à esquerda. Pode ser o caso de, por exemplo, números convertidos para cadeias de caracteres através da função STR().

LTRIM() é relacionada a RTRIM(), a qual remove espaços em branco à direita, e a ALLTRIM(), que remove espaços tanto à esquerda quanto à direita.

O contrário de ALLTRIM(), LTRIM(), e RTRIM() são as funções PADC(), PADR(), e PADL(), as quais centralizam, alinham à direita, ou alinham à esquerda as cadeias de caracteres, através da inserção de caracteres de preenchimento.

- ☑ **Sintaxe: LTRIM (< cString >)**
- ☑ **Parâmetros:**

cString	<cString> é a cadeia de caracteres a ser copiada sem os espaços em branco à esquerda.
----------------	---

- ☑ **Retorno:**

Caracter	LTRIM() retorna uma cópia de <cString>, sendo que os espaços em branco à esquerda foram removidos. Caso <cString> seja uma cadeia de caracteres nula (""), ou toda composta de espaços em branco, LTRIM() retorna uma cadeia de caracteres nula ("").
-----------------	---

MATHC()

Função utilizada para realizar operações matemáticas com strings que contém um valor numérico. MATHC() realiza algumas operações matemáticas como: Soma, Subtração, Divisão, Multiplicação e Exponenciação.

A função irá retornar uma string contendo o resultado da operação matemática, com uma especificação de até 18 casas de precisão no número.

☑ **Sintaxe:** MATHC (< cNum1 > , < cOperacao > , < cNum2 >)

☑ **Parâmetros:**

cNum1	String contendo um valor numérico, representando o número no qual desejamos realizar uma operação.
cOperacao	Representa a string que indica a operação que desejamos realizar. Olhar na tabela para verificar quais valores devem ser informados aqui.
cNum2	String contendo um valor numérico, representando o número no qual desejamos realizar uma operação.

☑ **Retorno:**

Caracter	Retorna uma nova string contendo o resultado matemático da operação.
-----------------	--

OEMTOANSI()

Função que transforma uma string no Formato OEM / MS-DOS Text para uma string ANSI Text (formato do Windows).

Quando utilizamos um programa baseado no MS-DOS para alimentar uma base de dados , os acentos e caracteres especiais são gravados como texto OEM . Para tornar possível a correta visualização destes dados em uma interface Windows , utilizamos a função OemToAnsi() para realizar a conversão.

Ao utilizarmos um programa baseado no Windows para alimentar uma base de dados , o texto é capturado no formato ANSI Text . Caso este texto seja utilizado para alimentar uma base de dados a ser acessada através de um programa MS-DOS , devemos converter o dado para OEM antes de gravá-lo , através da função AnsiToOem().

☑ **Sintaxe:** OemToAnsi (< cStringOEM >)

☑ **Parâmetros:**

cStringOEM	String em formato OEM - MsDos a ser convertida.
-------------------	---

☑ **Retorno:**

Caracter	String convertida para ser exibida no Windows (Formato ANSI).
-----------------	---

PADL() / PADR() / PADC()

Funções de tratamento de strings que inserem caracteres de preenchimento para completar um tamanho previamente especificado em vários formatos como data ou numéricos.

- ☐ PADC() centraliza <cExp>, adicionando caracteres de preenchimento à direita e à esquerda.
- ☐ PADL() adiciona caracteres de preenchimento à esquerda.
- ☐ PADR() adiciona caracteres de preenchimento à direita.

Caso o tamanho de <cExp> exceda o argumento <nTamanho>, todas as funções PAD() truncam string preenchida ao <nTamanho> especificado.

PADC(), PADL(), e PADR() são utilizadas para exibir cadeias de caracteres de tamanho variável em uma área de tamanho fixo. Elas podem ser usadas, por exemplo, para assegurar o alinhamento com comandos ?? consecutivos. Outra utilização é exibir textos em uma tela de tamanho fixo, para certificar-se de que o texto anterior foi completamente sobrescrito.

PADC(), PADL(), e PADR() são o contrário das funções ALLTRIM(), LTRIM(), e RTRIM(), as quais eliminam espaços em branco à esquerda e à direita de cadeias de caracteres.

- ☒ **Sintaxe: PADL / PADR / PADC (< cExp > , < nTamanho > , [cCaracPreench])**
- ☒ **Parâmetros**

cExp	Caractere, data, ou numérico no qual serão inseridos caracteres de preenchimento.
nTamanho	Tamanho da cadeia de caracteres a ser retornada.
cCaracPreench	Caractere a ser inserido em cExp. Caso não seja especificado, o padrão é o espaço em branco.

- ☒ **Retorno:**

Caracter	Retornam o resultado de <cExp> na forma de uma cadeia de caracteres preenchida com <cCaracPreench>, para totalizar o tamanho especificado por <nTamanho>.
-----------------	---

RAT()

Retorna a última posição de um caracter ou string dentro de outra string especificada.

- ☒ **Sintaxe: RAT(cCaractere, cString)**
- ☒ **Parâmetros**

cCaractere	Caractere ou string que se deseja verificar
cString	String na qual será verificada a existência do conteúdo de cCaractere.

REPLICATE()

A função Replicate() é utilizada para gerar uma cadeia de caracteres repetidos a partir de um caracter base informado, podendo a string gerada conter até 64KB. Caso seja especificado no parâmetro de itens a repetir o número zero, será retornada uma string vazia.

☑ **Sintaxe:** REPLICATE(cString, nCount)

☑ **Parâmetros:**

cString	Caracter que será repetido
nCount	Quantidade de ocorrências do caracter base que serão geradas na string de destino.

☑ **Retorno:**

cReplicated	String contendo as ocorrências de repetição geradas para o caracter informado.
--------------------	--

RTRIM()

Função para tratamento de caracteres utilizada para formatar cadeias de caracteres que contenham espaços em branco à direita. Ela é útil quando você deseja eliminar espaços em branco à direita ao se concatenar cadeias de caracteres. É o caso típico com campos de banco de dados que são armazenados em formato de tamanho fixo. Por exemplo, você pode usar RTRIM() para concatenar o primeiro e o último campos de nome para formar uma cadeia de caracteres de nome.

LTRIM() é relacionada a RTRIM(), que remove espaços em branco à direita, e a ALLTRIM(), que remove espaços em branco à direita e à esquerda.

O contrário de ALLTRIM(), LTRIM(), e RTRIM() são as funções PADC(), PADR(), e PADL(), as quais centralizam, alinham à direita, ou alinham à esquerda cadeias de caracteres, inserindo caracteres de preenchimento.

☑ **Sintaxe:** RTRIM (< cString >) --> cTrimString

☑ **Parâmetros:**

cString	<cString> é a cadeia de caracteres a ser copiada sem os espaços em branco à direita.
----------------	--

☑ **Retorno:**

Caracter	RTRIM() retorna uma cópia de <cString>, sendo que os espaços em branco à direita foram removidos. Caso <cString> seja uma cadeia de caracteres nula (""), ou totalmente composta por espaços, RTRIM() retorna uma cadeia de caracteres nula ("").
-----------------	---

SPACE()

Função de tratamento de caracteres utilizada para retornar uma quantidade especificada de espaços. A utilização desta função tem o mesmo efeito que REPLICATE(' ', <nCont>), e é normalmente utilizada para inicializar uma variável do tipo caractere, antes que a mesma seja associada a um GET.

Sintaxe: SPACE (< nCont >)

☒ **Parâmetros:**

nCont	A quantidade de espaços a serem retornados, sendo que o número máximo é 65.535 (64K).
--------------	---

☒ **Retorno:**

Caracter	Retorna uma cadeia de caracteres. Se <nCont> for zero, SPACE() retorna uma cadeia de caracteres nula ("").
-----------------	--

STRtokARR()

Função utilizada para retornar um array, de acordo com os dados passados como parâmetro para a função. Esta função recebe uma string <cValue> e um caractere <cToken> que representa um separador, e para toda ocorrência deste separador em <cValue> é adicionado um item no array.

☒ **Sintaxe:** STRtokARR (< cValue > , < cToken >)

☒ **Parâmetros:**

cValue	Representa a cadeia de caracteres no qual desejamos separar de acordo com <cToken>.
cToken	Representa o caractere que indica o separador em <cValue>.

☒ **Retorno:**

Array	Array de caracteres que representa a string passada como parâmetro.
--------------	---

Exemplo:

```
STRtokARR('1;2;3;4;5', ';')    //retorna {'1','2','3','4','5'}
```

STRTRAN()

Função utilizada para realizar a busca da ocorrência da string, sendo case sensitive.

☑ **Sintaxe:** STRTRAN (< cString > , < cSearch > , [cReplace] , [nStart] , [nCount])

☑ **Parâmetros:**

cString	Seqüência de caracteres ou campo memo a ser pesquisado.
cSearch	Seqüência de caracteres a ser procurada em cString.
cReplace	Seqüência de caracteres que deve substituir a string cSearch. Caso não seja especificado, as ocorrências de cSearch em cString serão substituídas por uma string nula ("").
nStart	nStart corresponde ao número seqüencial da primeira ocorrência de cSearch em cString a ser substituída por cReplace. Se este argumento for omitido , o default é 1 (um) . Caso seja passado um numero menor que 1, a função retornará uma string em branco ("").
nCount	nCount corresponde ao número máximo de trocas que deverá ser realizada pela função . Caso este argumento não seja especificado , o default é substituir todas as ocorrências encontradas.

☑ **Retorno:**

Code-Block	A função STRTRAN retorna uma nova string, com as ocorrências especificadas de cSearch trocadas para cReplace, conforme parametrização.
-------------------	--

STUFF()

Função que permite substituir um conteúdo caractere em uma string já existente, especificando a posição inicial para esta adição e o número de caracteres que serão substituídos.

☑ **Sintaxe:** STUFF(cString, nPosI nicial, nExcluir, cAdicao)

☑ **Parâmetros:**

cString	A cadeia de caracteres destino na qual serão eliminados e inseridos caracteres.
nPosI nicial	A posição inicial na cadeia de caracteres destino onde ocorre a inserção/eliminação.
nExcluir	A quantidade de caracteres a serem eliminados.
cAdicao	A cadeia de caracteres a ser inserida.

☑ **Retorno:**

Caracter	Retorna a nova string gerada pela função com as modificações.
-----------------	---

Exemplo:

```

cLin := Space(100)+cEOL // Cria a string base
cCpo := PADR(SA1->A1_FILIAL,02) // Informação que será armazenada na string
cLin := Stuff(cLin,01,02,cCpo) // Substitui o conteúdo de cCpo na string base

```

SUBSTR()

Retorna parte do conteúdo de uma string especificada, de acordo com a posição inicial deste conteúdo na string e a quantidade de caracteres que deverá ser retornada a partir daquele ponto (inclusive).

☑ **Sintaxe: SUBSTR(cString, nPosInicial, nCaracteres)**

☑ **Parâmetros**

cString	String que se deseja verificar
nPosInicial	Posição inicial da informação que será extraída da string
nCaracteres	Quantidade de caracteres que deverá ser retornada a partir daquele ponto (inclusive).

Exemplo:

```

cCampo := "A1_NOME"
nPosUnder := AT(cCampo)
cPrefixo := SUBSTR(cCampo,1, nPosUnder) // → "A1_"

```

TRANSFORM()

Função de conversão que formata valores caractere, data, lógicos e numéricos conforme um string de máscara especificado, a qual inclui uma combinação de strings de template e funções de picture. Ela faz o mesmo que a cláusula PICTURE do comando @...SAY, sendo normalmente utilizada para formatar dados a serem enviados à tela ou à impressora.

☑ **Sintaxe: TRANSFORM (< cExp > , < cSayPicture >)**

☑ **Parâmetros:**

cExp	O valor a ser formatado. Esta expressão pode ser qualquer tipo de dados válidos, exceto vetor, bloco de código, e NIL.
cSayPicture	Uma string de caracteres de máscara e template usado para descrever o formato da cadeia de caracteres a ser retornada.

☑ **Retorno:**

-	Retorna a conversão de <cExp> para uma cadeia de caracteres formatada conforme a definição em <cSayPicture>.
---	--

UPPER()

Retorna uma string com todos os caracteres maiúsculos, tendo como base a string passada como parâmetro.

- ☒ **Sintaxe: UPPER(cString)**
- ☒ **Parâmetros**

cString	String que será convertida para caracteres maiúsculos.
----------------	--

Exemplo:

```
cTexto := "ADVPL"
```

```
MSGINFO("Texto:" + LOWER(cTexto))
```



Anotações

Manipulação de data / hora

CDOW()

Função que converte uma data para uma cadeia de caracteres.

- ☒ **Sintaxe:** CDOW(*dExp*)
- ☒ **Parâmetros:**

dExp	Data que será convertida.
-------------	---------------------------

- ☒ **Retorno:**

cDayWeek	Nome do dia da semana como uma cadeia de caracteres. A primeira letra é maiúscula e as demais minúsculas.
-----------------	---

Exemplo:

```
dData := DATE() // Resultado: 09/01/90
cDiaDaSemana := CDOW( DATE() ) // Resultado: Friday
cDiaDaSemana := CDOW( DATE() + 7 ) // Resultado: Friday
cDiaDaSemana := CDOW( CTOD( "06/12/90" ) ) // Resultado: Tuesday
```

CMONTH()

Função de conversão de datas que retorna uma cadeia de caracteres com o nome do mês em inglês.

- ☒ **Sintaxe:** CMONTH(*dData*)
- ☒ **Parâmetros:**

dData	Data que será convertida.
--------------	---------------------------

- ☒ **Retorno:**

cMonth	Retorna o nome do mês em uma cadeia de caracteres. A primeira letra do retorno em maiúscula e o restante do nome, em minúsculas.
---------------	--

Exemplo:

```
cMes := CMONTH( DATE() ) // Resultado: September
cMes := CMONTH( DATE() + 45 ) // Resultado: October
cMes := CMONTH( CTOD( "12/01/94" ) ) // Resultado: December
cMes := SUBSTR( CMONTH( DATE() ), 1, 3 ) + STR( DAY( DATE() ) ) // Resultado: Sep 1
```

DATE()

Função que retorna a data do atual sistema. O formato de saída é controlado pelo comando SET DATE, sendo que o formato padrão é mm/dd/yy.

☒ **Sintaxe:** DATE()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

dData	Data do sistema.
-------	------------------

Exemplo:

```
dData := DATE() // Resultado: 09/01/01
dData := DATE() + 30 // Resultado: 10/01/01
dData := DATE() - 30 // Resultado: 08/02/90
dData := DATE()
cMes := CMONTH(dData) // Resultado: September
```

DAY()

Função de conversão de datas usada para converter o valor data em um número inteiro que representa o dia do mês. Esta função pode ser usada em conjunto com CMONTH() e YEAR() para formatar datas. Pode ser usada também em diversos cálculos envolvendo datas.

☒ **Sintaxe:** DAY(dData)

☒ **Parâmetros:**

dData	Data que será convertida.
-------	---------------------------

☒ **Retorno:**

nDias	Se o mês do argumento dData for fevereiro, anos bissextos são considerados. Se a data do argumento dData for 29 de fevereiro e o ano não for bissexto, ou se o argumento dData for vazio.
-------	---

Exemplo:

// Estes exemplos mostram a função DAY() de diversas maneiras:

```
dData := DATE() // Resultado: 09/01/01
```

```
nDia := DAY(DATE()) // Resultado: 1
```

```
nDia := DAY(DATE()) + 1 // Resultado: 2
```

```
nDia := DAY(CTOD("12/01/94")) // Resultado: 1
```

// Este exemplo mostra a função DAY() usada em conjunto com CMONTH() e

YEAR() para formatar o valor da data:

```
dData := Date()
```

```
cData := CMONTH(dData) + STR(DAY(dData)) + "," + STR(YEAR(dData)) // Resultado: June 15, 2001
```

DOW()

Função que converte uma data para o valor numérico que representa o dia da semana. Útil quando se deseja fazer cálculos semanais. DOW() é similar a CDOW(), que retorna o dia da semana como uma cadeia de caracteres.

✓ **Sintaxe:** DOW(*dData*)

✓ **Parâmetros:**

dData	Data que será convertida.
--------------	---------------------------

✓ **Retorno:**

nDia	Retorna um número entre zero e sete, representando o dia da semana. O primeiro dia da semana é 1 (Domingo) e o último é 7 (Sábado). Se a data for vazia ou inválida, DOW() retorna zero.
-------------	--

Exemplo:

```
dData := DATE() // Resultado: 09/01/01
nDiaDaSemana := DOW( DATE() ) // Resultado: 3
cDiaDaSemana := CDOW( DATE() ) // Resultado: Tuesday
nDiaDaSemana := DOW( DATE() - 2 ) // Resultado: 1
cDiaDaSemana := CDOW( DATE() - 2 ) // Resultado: Sunday
```

DTOC()

Função para conversão de uma data para uma cadeia de caracteres formatada segundo o padrão corrente, definido pelo comando SET DATE. Se for necessária a utilização de formatação especial, use a função TRANSFORM().

Em expressões de índices de arquivo, use DTOS() no lugar de DTOC() para converter datas para cadeia de caracteres.

✓ **Sintaxe:** DTOC(*dData*)

✓ **Parâmetros:**

dData	Data que será convertida.
--------------	---------------------------

✓ **Retorno:**

cData	É uma cadeia de caracteres representando o valor da data. O retorno é formatado utilizando-se o formato corrente definido pelo comando SET DATE FORMAT. O formato padrão é mm/dd/yy. Para uma data nula ou inválida, o retorno será uma cadeia de caracteres com espaços e tamanho igual ao formato atual.
--------------	--

Exemplo:

```
cData := DATE() // Resultado: 09/01/90
cData := DTOC( DATE() ) // Resultado: 09/01/90
cData := "Today is " + DTOC( DATE() ) // Resultado: Today is 09/01/90
```


DTOS()

Função para conversão de uma data que pode ser usada para criar expressões de índice. O resultado é estruturado visando manter a ordem correta do índice (ano, mês, dia).

✓ **Sintaxe:** **DTOS(*dData*)**

✓ **Parâmetros:**

dData	Data que será convertida.
--------------	---------------------------

✓ **Retorno:**

sData	Retorna uma cadeia de caracteres com oito byte de tamanho no formato <code>yyyymmdd</code> . Quando <i>dData</i> é nulo ou invalido, DTOS() retorna uma cadeia de caracteres com oito espaços. O valor retornado não é afetado pela formato da data corrente.
--------------	---

Exemplo:

```
cData := DATE() // Resultado: 09/01/90
cData := DTOS( DATE() ) // Resultado: 19900901
nLen := LEN( DTOS( CTOD( "" ) ) ) // Resultado: 8
```

ELAPTIME()

Função que retorna uma cadeia de caracteres contendo a diferença de tempo no formato `hh:mm:ss`, onde hh é a hora (1 a 24), mm os minutos e ss os segundos.

✓ **Sintaxe:** **ElapTime(*cHoraInicial* , *cHoraFinal*)**

✓ **Parâmetros:**

cHoraInicial	Informe a hora inicial no formato <code>hh:mm:ss</code> , onde hh é a hora (1 a 24), mm os minutos e ss os segundos
CHoraFinal	Informe a hora final no formato <code>hh:mm:ss</code> , onde hh é a hora (1 a 24), mm os minutos e ss os segundos.

✓ **Retorno:**

Caracter	A diferença de tempo no formato <code>hh:mm:ss</code> , onde hh é a hora (1 a 24), mm os minutos e ss os segundos.
-----------------	--

Exemplo:

```
cHoraInicio := TIME() // Resultado: 10:00:00
...
<instruções>
...
cElapsed := ELAPTIME( TIME(), cHoraInicio )
```

MONTH()

Função de conversão que extrai da data o valor numérico do mês, semelhante a função que retorna o nome do mês a partir do valor de dData.

✓ **Sintaxe:** MONTH(dData)

✓ **Parâmetros:**

dData	Data que será convertida.
--------------	---------------------------

✓ **Retorno:**

Numérico	>=0 e <=12 → Para uma data válida. 0 → Se a data for nula ou inválida
-----------------	--

Exemplo:

```
dData := DATE() // Resultado: 09/01/01  
nMes := MONTH(DATE()) // Resultado: 9  
nMes := MONTH(DATE()) + 1 // Resultado: 10
```

SECONDS()

Esta função retorna o número de segundos decorridos desde a meia-noite, segundo a hora do sistema. Está relacionada à função TIME() que retorna a hora do sistema como uma cadeia de caracteres no formato hh:mm:ss.

✓ **Sintaxe:** SECONDS()

✓ **Parâmetros:**

Nenhum	.
---------------	---

✓ **Retorno:**

Numérico	>=0 e <=86399 → Retorna a hora do sistema em segundos. O valor numérico representa o número de segundos decorridos desde a meia-noite, baseado no relógio de 24 horas e varia de 0 a 86399.
-----------------	---

Exemplo:

```
cHora := TIME() // Resultado: 10:00:00  
cSegundos := SECONDS() // Resultado: 36000.00  
  
//Este exemplo usa a função SECONDS() para cronometrar o tempo decorrido:  
  
LOCAL nStart, nElapsed  
nStart:= SECONDS()
```

TIME()

Função que retorna a hora do sistema como uma cadeia de caracteres, e que está relacionada com SECONDS(), que retorna o valor inteiro representando o número de segundos desde a meia-noite. SECONDS() é geralmente usada no lugar de TIME() para cálculos.

☑ **Sintaxe:** TIME()

☑ **Parâmetros:**

Nenhum	.
--------	---

☑ **Retorno:**

Caracter	A hora do sistema como uma cadeia de caracteres no formato hh:mm:ss onde hh é a hora (1 a 24), mm os minutos e ss os segundos.
----------	--

Exemplo:

```
cTime := TIME() // Resultado: 10:37:17
cHora := SUBSTR(cTime, 1, 2) // Resultado: 10
cMinutos := SUBSTR(cTime, 4, 2) // Resultado: 37
cSegundos := SUBSTR(cTime, 7, 2) // Resultado: 17
```

YEAR()

YEAR() é uma função de conversão de data que extrai o valor numérico do ano. YEAR() é membro de um grupo de funções que retornam valores numéricos de uma data. O grupo inclui DAY() e MONTH() que retornam o dia e o mês como valores numéricos.

☑ **Sintaxe:** YEAR(dData)

☑ **Parâmetros:**

dData	Data que será convertida.
-------	---------------------------

☑ **Retorno:**

Numérico	Valor numérico do ano da data especificada em dData incluindo os dígitos do século. O valor retornado não é afetado pelos valores especificados pelos comandos SET DATE ou SET CENTURY.
----------	---

Para uma data inválida ou nula será retornado o valor 0.

Exemplo 01:

```
dData := DATE() // Resultado: 09/20/01
dAno := YEAR(dData) // Resultado: 2001
dAno := YEAR(dData) + 11 // Resultado: 2012
```

Exemplo 02:

```
// Este exemplo cria uma função de usuário que usa a função YEAR() para formatar o valor da  
// data:
```

```
cData := Mdy( DATE() ) // Result: September 20, 1990  
FUNCTION Mdy( dDate )  
RETURN CMONTH(dDate) + " " + LTRIM(STR(DAY(dDate))) + "," + STR(YEAR(dDate))
```

Manipulação de variáveis numéricas

ABS()

Retorna um valor absoluto (independente do sinal) com base no valor especificado no parâmetro.

- ☒ **Sintaxe: ABS(nValor)**
- ☒ **Parâmetros**

nValor	Valor que será avaliado
---------------	-------------------------

Exemplo:

```
nPessoas := 20  
nLugares := 18  
  
IF nPessoas < nLugares  
    MSGINFO("Existem "+CVALTOCHAR(nLugares- nPessoas)+"disponíveis")  
ELSE  
    MSGSTOP("Existem "+CVALTOCHAR(ABS(nLugares- nPessoas))+"faltando")  
ENDIF
```

ALEATORIO()

Gera um número aleatório de acordo com a semente passada. Esta função retorna um número aleatório menor ou igual ao primeiro parâmetro informado, usando como semente o segundo parâmetro. É recomendado que esta semente seja sempre o último número aleatório gerado por esta função.

- ☒ **Sintaxe: Aleatorio(nMax,nSeed)**
- ☒ **Parâmetros**

nMax	Número máximo para a geração do número aleatório
nSeed	Semente para a geração do número aleatório

Exemplo – Função ALEATORIO()

```
nSeed := 0
For i := 1 to 100
nSeed := Aleatorio(100,nSeed)
? Str(i,3)+"§ numero aleatorio gerado: "+Str(nSeed,3)
Next i
inkey(0)
Return
```

INT()

Retorna a parte inteira de um valor especificado no parâmetro.

☑ **Sintaxe: INT(nValor)**

☑ **Parâmetros**

nValor	Valor que será avaliado
---------------	-------------------------

Exemplo:

```
STATIC FUNCTION COMPRAR(nQuantidade)

LOCAL nDinheiro := 0.30
LOCAL nPrcUnit := 0.25

IF nDinheiro >= (nQuantidade*nPrcUnit)
    RETURN nQuantidade
ELSEIF nDinheiro > nPrcUnit
    nQuantidade := INT(nDinheiro / nPrcUnit)
ELSE
    nQuantidade := 0
ENDIF

RETURN nQuantidade
```

NOROUND()

Retorna um valor, truncando a parte decimal do valor especificado no parâmetro de acordo com a quantidade de casas decimais solicitadas.

☑ **Sintaxe: NOROUND(nValor, nCasas)**

☑ **Parâmetros**

nValor	Valor que será avaliado
nCasas	Número de casas decimais válidas. A partir da casa decimal especificada os valores serão desconsiderados.

Exemplo – Função NOROUND()

nBase := 2.985

nValor := NOROUND(nBase,2) → 2.98

RANDOMIZE()

Através da função RANDOMIZE() , geramos um numero inteiro aleatório, compreendido entre a faixa inferior e superior recebida através dos parâmetros nMinimo e nMaximo, respectivamente.

Observação :

- ❑ O limite inferior recebido através do parâmetro nMinimo é "maior ou igual a ", podendo ser sorteado e fazer parte do retorno; porém o limite superior é "menor que", de modo a nunca será atingido ou devolvido no resultado. Por exemplo , a chamada da função RANDOMIZE(1,2) sempre retornará 1 .

☑ **Sintaxe:** RANDOMIZE (< nMinimo > , < nMaximo >)

☑ **Parâmetros**

nMinimo	Corresponde ao menor numero a ser gerado pela função.
nMaximo	Corresponde ao maior número (menos um) a ser gerado pela função.

☑ **Retorno:**

Numérico	Numero randômico , compreendido no intervalo entre (nMinimo) e (nMaximo-1) : O numero gerado pode ser maior ou igual à nMinimo e menor ou igual a nMaximo-1 .
-----------------	---

ROUND()

Retorna um valor, arredondando a parte decimal do valor especificado no parâmetro de acordo com a quantidade de casas decimais solicitadas, utilizando o critério matemático.

☑ **Sintaxe:** ROUND(nValor, nCasas)

☑ **Parâmetros**

nValor	Valor que será avaliado
nCasas	Número de casas decimais válidas. As demais casas decimais sofrerão o arredondamento matemático, aonde: Se $nX \leq 4 \rightarrow 0$, senão +1 para a casa decimal superior.

Exemplo:

nBase := 2.985

nValor := ROUND(nBase,2) → 2.99

Manipulação de arquivos

ADIR()

Função que preenche os arrays passados com os dados dos arquivos encontrados, através da máscara informada. Tanto arquivos locais (Remote) como do servidor podem ser informados.

Importante: *ADir é uma função obsoleta, utilize sempre Directory().*

☑ **Sintaxe:** **ADIR([cArqEspec], [aNomeArq], [aTamanho], [aData], [aHora], [aAtributo])**

☑ **Parâmetros:**

cArqEspec	Caminho dos arquivos a serem incluídos na busca de informações. Segue o padrão para especificação de arquivos, aceitando arquivos no servidor Protheus e no Cliente. Caracteres como * e ? são aceitos normalmente. Caso seja omitido, serão aceitos todos os arquivos do diretório default (*.*).
aNomeArq	Array de Caracteres. É o array com os nomes dos arquivos encontrados na busca. O conteúdo anterior do array é apagado.
aTamanho	Array Numérico. São os tamanhos dos arquivos encontrados na busca.
aData	Array de Datas. São as datas de modificação dos arquivos encontrados na busca.
aHora	Array de Caracteres. São os horários de modificação dos arquivos encontrados. Cada elemento contém horário no formato: hh:mm:ss.
aAtributos	Array de Caracteres. São os atributos dos arquivos, caso esse array seja passado como parâmetros, serão incluídos os arquivos com atributos de sistema e ocultos.

☑ **Retorno:**

nArquivos	Quantidade de arquivos encontrados.
------------------	-------------------------------------

Exemplo:

```
LOCAL aFiles[ADIR("*.TXT")]
ADIR("*.TXT", aFiles)
AEVAL(aFiles, { |element| QOUT(element) })
```



Anotações

CGETFILE()

Função utilizada para seleção de um arquivo ou diretório, disponibilizando uma interface gráfica para amigável para o usuário. Esta função está normalmente associada ao recurso de abrir ou salvar arquivos, permitindo para esta última a digitação opcional do nome do arquivo que será gravado.

☑ **Sintaxe:** `cGetFile (ExpC1, ExpC2, ExpN1, ExpC3, ExpL1, ExpN2,ExpL2)`

☑ **Parâmetros:**

ExpC1	Mascara para filtro (Ex: 'Informes Protheus (*.###R) *.###R')
ExpC2	Título da Janela
ExpN1	Numero da mascara default (Ex: 1 p/ *.exe)
ExpC3	Diretório inicial se necessário
ExpL1	.T. para mostrar botão como 'Salvar' e .F. para botão 'Abrir'
ExpN2	Mascara de bits para escolher as opções de visualização do Objeto.
ExpL2	.T. para exibir diretório [Servidor] e .F. para não exibir

☑ **Máscaras de bits para opções:**

GETF_OVERWRITEPROMPT	Solicita confirmação para sobrescrever
GETF_MULTISELECT	Permite selecionar múltiplos arquivos
GETF_NOCHANGEDIR	Não permite mudar o diretório inicial
GETF_LOCALFLOPPY	Exibe o(s) Drive(s) de disquete da maquina local
GETF_LOCALHARD	Exibe o(s) HardDisk(s) Local(is)
GETF_NETWORKDRIVE	Exibe os drives da rede (Mapeamentos)
GETF_SHAREWARE	Não implementado
GETF_RETDIRECTORY	Retorna um diretório

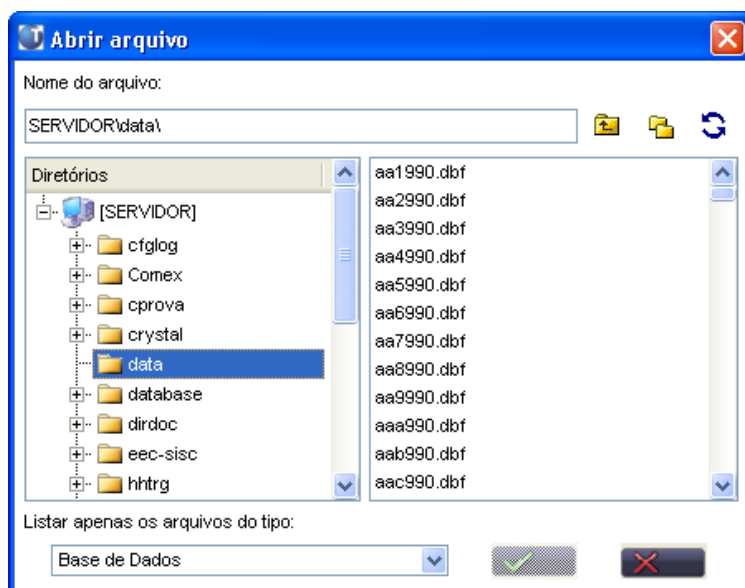
Exemplo:

```
cGetFile ( '*.PRW|*.CH' , 'Fontes', 1, 'C:\VER507', .F., GETF_LOCALHARD +  
GETF_LOCALFLOPPY)
```



Anotações

☑ **Aparência:**



Dica

Para permitir a seleção de diversos arquivos contidos em um diretório é necessário combinar as funções CGETFILE(), DIRECTORY() e o objeto LISTBOX() conforme abaixo:

- CGETFILE: exibe os diretórios disponíveis e retorna o nome do item selecionado.
- DIRECTORY: efetua a leitura dos arquivos contidos no diretório retornado pela CGETFILE.
- LISTBOX: Exibe uma tela de seleção de com a opção de marcação, para que sejam selecionados os arquivos que serão processados.



Anotações

Função Principal: SELFIE()

```
#include "protheus.ch"

//+-----+
//| Rotina | SELFIE | Autor | ARNALDO R. JUNIOR | Data | 01.01.2007 |
//+-----+
//| Descr. | Função exemplo para seleção de múltiplos arquivos. |
//+-----+
//| Uso    | CURSO DE ADVPL |
//+-----+

USER FUNCTION SELFIE()

LOCAL cDirectory := ""
LOCAL aArquivos  := {}
LOCAL nArq       := 0

PRIVATE aParamFile:= ARRAY(1)

IF !PARBOXFILE()
    RETURN
ENDIF

// Exibe a estrutura de diretório e permite a seleção dos arquivos que serão
// processados
cDirectory := ALLTRIM(cGetFile("Arquivos de Dados|" + aParamFile[1] + "|" ,
'Importação de lançamentos', 0, '', .T., GETF_OVERWRITEPROMPT + GETF_NETWORKDRIVE
+ GETF_RETDIRIRECTORY,.T.))
aArquivos  := Directory(cDirectory+"*.*")
aArquivos  := MARKFILE(aArquivos,cDirectory,aParamFile[1],@lSelecao)

FOR nArq TO Len(aArquivos)

    IF !aArquivos[nArq][1]
        LOOP
    ENDIF

    <...processamento...>

NEXT nArq

RETURN
```

Função auxiliar: PARBOXFILE()

```
//+-----+
//| Rotina | PARBOXFILE | Autor | ARNALDO R. JUNIOR Data | 01.01.2007 |
//+-----+
//| Descr. | Função exemplo de uso da PARAMBOX em conjunto com CGETFILE|
//+-----+
//| Uso    | CURSO DE ADVPL                                     |
//+-----+

STATIC FUNCTION PARBOXFILE()

Local aParamBox := {}
Local cTitulo   := "Filtros Adicionais"
Local aRet      := {}
Local bOk       := {|| .T.}
Local aButtons  := {}
Local lCentered := .T.
Local nPosx
Local nPosy
Local cLoad := ""
Local lCanSave := .F.
Local lUserSave := .F.
Local nX      := 0
Local lRet    := .T.

AADD(aParamBox,{2,"Tipo de arquivo"
,2,{"*.dbf","*.dtt"},100,"AlwaysTrue()",.T.})

lRet := ParamBox(aParamBox, cTitulo, aRet, bOk, aButtons, lCentered, nPosx,
nPosy,, cLoad, lCanSave, lUserSave)

IF ValType(aRet) == "A" .AND. Len(aRet) == Len(aParamBox)
    For nX := 1 to Len(aParamBox)
        If aParamBox[nX][1] == 1
            aParaml02[nX] := aRet[nX]
        ElseIf aParamBox[nX][1] == 2 .AND. ValType(aRet[nX]) == "C"
            aParaml02[nX] := aRet[nX] // Tipo do arquivo
        ElseIf aParamBox[nX][1] == 2 .AND. ValType(aRet[nX]) == "N"
            aParaml02[nX] := aParamBox[nX][4][aRet[nX]] // Tipo do arquivo
        Endif
    Next nX
ENDIF

RETURN lRet
```

Função auxiliar: MARKFILE()

```
//+-----+
//| Rotina | MARKFILE | Autor | ARNALDO R. JUNIOR | Data | 01.01.2007 |
//+-----+
//| Descr. | Função exemplo para marcação de múltiplos arquivos. |
//+-----+
//| Uso    | CURSO DE ADVPL |
//+-----+

STATIC FUNCTION MARKFILE(aArquivos,cDiretorio,cDriver,lSelecao)

Local aChaveArq := {}
Local cTitulo   := "Arquivos para importação: "
Local bCondicao := {||.T.}
// Variáveis utilizadas na seleção de categorias
Local oChkQual,lQual,oQual,cVarQ
// Carrega bitmaps
Local oOk      := LoadBitmap( GetResources(), "LBOK")
Local oNo      := LoadBitmap( GetResources(), "LBNO")
// Variáveis utilizadas para lista de filiais
Local nx       := 0
Local nAchou   := 0

//+-----+
//| Carrega os arquivos do diretório no array da ListBox |
//+-----+
For nx := 1 to Len(aArquivos)
    //+-----+
    //| aChaveArq - Contem os arquivos que serão exibidos para seleção |
    //+-----+
    AADD(aChaveArq,{.F.,aArquivos[nx][1],cDiretorio})
Next nx

//+-----+
//| Monta tela para seleção dos arquivos contidos no diretório |
//+-----+
DEFINE MSDIALOG oDlg TITLE cTitulo STYLE DS_MODALFRAME From 145,0 To 445,628;
OF oMainWnd PIXEL
oDlg:lEscClose := .F.
@ 05,15 TO 125,300 LABEL UPPER(cDriver) OF oDlg PIXEL
@ 15,20 CHECKBOX oChkQual VAR lQual PROMPT "Inverte Seleção" SIZE 50, 10;
OF oDlg PIXEL;
ON CLICK (AEval(aChaveArq, {|z| z[1] := If(z[1]=.T.,.F.,.T.)}),;
oQual:Refresh(.F.))
@ 30,20 LISTBOX oQual VAR cVarQ Fields HEADER "", "Código", "Descrição" SIZE;
273,090 ON DBLCLICK (aChaveArq:=Troca(oQual:nAt,aChaveArq),oQual:Refresh());
NoScroll OF oDlg PIXEL
oQual:SetArray(aChaveArq)
oQual:bLine := { || {If(aChaveArq[oQual:nAt,1],oOk,oNo),;
aChaveArq[oQual:nAt,2],aChaveArq[oQual:nAt,3]}}
DEFINE SBUTTON FROM 134,240 TYPE 1 ACTION IIF(MarcaOk(aChaveArq),;
(lSelecao := .T., oDlg:End(),.T.),.F.) ENABLE OF oDlg
DEFINE SBUTTON FROM 134,270 TYPE 2 ACTION (lSelecao := .F., oDlg:End());
ENABLE OF oDlg
ACTIVATE MSDIALOG oDlg CENTERED

RETURN aChaveArq
```

Função auxiliar: TROCA()

```
//+-----+
//| Rotina | TROCA      | Autor | ARNALDO R. JUNIOR | Data | 01.01.2007 |
//+-----+
//| Uso    | CURSO DE ADVPL |
//+-----+

STATIC FUNCTION Troca(nIt,aArray)
aArray[nIt,1] := !aArray[nIt,1]
Return aArray
```

Função auxiliar: MARCAOK()

```
//+-----+
//| Rotina | MARCAOK   | Autor | ARNALDO R. JUNIOR | Data | 01.01.2007 |
//+-----+
//| Uso    | CURSO DE ADVPL |
//+-----+

STATIC FUNCTION MarcaOk(aArray)
Local lRet:=.F.
Local nx:=0

// Checa marcações efetuadas
For nx:=1 To Len(aArray)
    If aArray[nx,1]
        lRet:=.T.
    EndIf
Next nx
// Checa se existe algum item marcado na confirmação
If !lRet
    HELP("SELFIL",1,"HELP","SEL. FILE","Não existem itens marcados",1,0)
EndIf

Return lRet
```



Anotações

CPYS2T()

Função utilizada para copiar um arquivo do servidor para o cliente (Remote), sendo que os caracteres "*" e "?" são aceitos normalmente. Caso a compactação seja habilitada (*ICompacta*), os dados serão transmitidos de maneira compacta e descompactados antes do uso.

☒ **Sintaxe:** CPYS2T (< cOrigem > , < cDestino > , [ICompacta])

☒ **Parâmetros:**

cOrigem	Nome(s) dos arquivos a serem copiados, aceita apenas arquivos no servidor, WildCards (* e ?) são aceitos normalmente.
cDestino	Diretório com o destino dos arquivos no Client (Remote).
ICompacta	Indica se a cópia deve ser feita compactando o arquivo antes do envio.

☒ **Retorno:**

Lógico	ISucess retorna .T. caso o arquivo seja copiado com sucesso , ou .F. em caso de falha na cópia.
---------------	---

Exemplo:

```
// Copia arquivos do servidor para o remote local, compactando antes de transmitir
CpyS2T( "\\BKP\MANUAL.DOC", "C:\TEMP", .T. )
// Copia arquivos do servidor para o remote local, sem compactar antes de transmitir
CpyS2T( "\\BKP\MANUAL.DOC", "C:\TEMP", .F. )
```

CPYT2S()

Função utilizada para copiar um arquivo do cliente (Remote) para o servidor, sendo que os caracteres "*" e "?" são aceitos normalmente. Caso a compactação seja habilitada (*ICompacta*), os dados serão transmitidos de maneira compacta e descompactados antes do uso.

☒ **Sintaxe:** CpyT2S(cOrigem, cDestino, [ICompacta])

☒ **Parâmetros:**

cOrigem	Nomes dos arquivos a serem copiados, aceita apenas arquivos locais (Cliente), WildCards são aceitos normalmente.
cDestino	Diretório com o destino dos arquivos no remote (Cliente).
ICompacta	Indica se a cópia deve ser feita compactando o arquivo antes.

☒ **Retorno:**

Lógico	Indica se o arquivo foi copiado para o cliente com sucesso.
---------------	---

Exemplo:

```
// Copia arquivos do cliente( remote ) para o Servidor compactando antes de transmitir
CpyT2S( "C:\TEMP\MANUAL.DOC", "\\BKP", .T. )
// Copia arquivos do cliente( remote ) para o Servidor sem compactar.
CpyT2S( "C:\TEMP\MANUAL.DOC", "\\BKP" )
```

CURDIR()

Função que retorna o diretório corrente do servidor. O caminho retornado é sempre relativo ao RootPath definido na configuração do Environment no .INI do Protheus Server. Inicialmente , o diretório atual da aplicação é o constante na chave StartPath , também definido na configuração do Environment no .INI do Protheus Server.

Caso seja passado o parâmetro cNovoPath , este path é assumido como sendo o Path atual. Caso o path recebido como parâmetro não exista , seja inválido , ou seja um path absoluto (iniciado com uma letra de drive ou caminho de rede), a função não irá setar o novo path, mantendo o atual .

☑ **Sintaxe:** CURDIR ([cNovoPath])

☑ **Parâmetros:**

cNovoPath	Caminho relativo , com o novo diretório que será ajustado como corrente.
------------------	--

☑ **Retorno:**

Caracter	Diretório corrente, sem a primeira barra.
-----------------	---

Exemplo:

```
cOldDir := curdir()
cNewDir := '\webadv\xis'
curdir(cNewDir) // Troca o path
If cNewDir <> '\' + curdir() // E verifica se trocou mesmo
  conout('Falha ao Trocar de Path de '+cOldDir + ' para '+cNewDir)
Else
  conout('Path de '+cOldDir + ' trocado para '+cNewDir+' com sucesso.')
Endif
```



Anotações

DIRECTORY()

Função de tratamento de ambiente que retorna informações a respeito dos arquivos no diretório corrente ou especificado. É semelhante a ADIR(), porém retorna um único vetor ao invés de adicionar valores a uma série de vetores existentes passados por referência.

DIRECTORY() pode ser utilizada para realizar operações em conjuntos de arquivos. Em combinação com AEVAL(), você pode definir um bloco que pode ser aplicado a todos os arquivos que atendam a <cDirSpec> especificada.

Para tornar as referências aos vários elementos de cada sub-vetor de arquivo mais legíveis, é fornecido o arquivo header Directry.ch, que contém os #defines para os subarray subscripts.

☑ TABELA A: Atributos de DIRECTORY()

Atributo	Significado
H	Incluir arquivos ocultos
S	Incluir arquivos de sistema
D	Incluir diretórios
V	Procura pelo volume DOS e exclui outros arquivos

Nota: Arquivos normais são sempre incluídos na pesquisa, a não ser que V seja especificado.

☑ TABELA B: Estrutura dos Subvetores de DIRECTORY()

Posição	Metasímbolo	Directry.ch
1	cNome	F_NAME
2	cTamanho	F_SIZE
3	dData	F_DATE
4	cHora	F_TIME
5	cAtributos	F_ATT

☑ Sintaxe: DIRECTORY (< cDirSpec > , [])

☑ Parâmetros:

cDirSpec	<cDirSpec> especifica o drive, diretório e arquivo para a pesquisa no diretório. Caracteres do tipo coringa são permitidos na especificação de arquivos. Caso <cDirSpec> seja omitido, o valor padrão é *.*. O caminho especificado pode estar na estação (remote) , ou no servidor, obedecendo às definições de Path Absoluto / Relativo de acesso.
cAtributos>	<cAtributos> especifica que arquivos com atributos especiais devem ser incluídos na informação retornada. <cAtributos> consiste em uma cadeia de caracteres que contém um ou mais dos seguintes caracteres, contidos na tabela adicional A , especificada anteriormente.

☒ **Retorno:**

Array	DIRECTORY() retorna um vetor de sub-vetores, sendo que cada sub-vetor contém informações sobre cada arquivo que atenda a <cDirSpec>.Veja maiores detalhes na Tabela B, discriminada anteriormente.
--------------	--

Exemplo:

```
#INCLUDE "Directry.ch"
```

```
aDirectory := DIRECTORY("*.","D")
AEVAL( aDirectory, { |aFile| CONOUT(aFile[F_NAME])} )
```

DIRREMOVE()

Função que elimina um diretório específico. Caso especifiquemos um path sem a unidade de disco , ele será considerado no ambiente do Servidor , a partir do RootPath do ambiente (caso o path comece com \), ou a partir do diretório corrente (caso o path não seja iniciado com \).

Quando especificado um path absoluto (com unidade de disco preenchida), a função será executada na estação onde está sendo executado o Protheus Remote. Quando executamos a função DirRemove() em JOB (processo isolado no Server , sem interface), não é possível especificar um Path absoluto de disco. Caso isto seja realizado , a função retornará .F. e FError() retornará -1 (Syntax Error).

Note que é necessário ter direitos suficientes para remover um diretório, e o diretório a ser eliminado precisa estar vazio, sem subdiretórios ou arquivos dentro do mesmo.

☒ **Sintaxe: DIRREMOVE (< cDiretorio >)**

☒ **Parâmetros:**

cDiretorio	Nome do diretório a ser removido.
-------------------	-----------------------------------

☒ **Retorno:**

Lógico	ISucesso será .T. caso o diretório tenha sido eliminado , ou .F. caso não seja possível excluir o diretório. Quando a função DirRemove retornar .F. , é possível obter mais detalhes da ocorrência recuperando o código do Erro através da função FError().
---------------	---

Exemplo:

```
cDelPath := 'c:\TmpFiles'
IRemoveOk := DIRREMOVE(cDelPath)

IF !IRemoveOk
  MsgStop('Falha ao remover a pasta '+cDelPath+' ( File Error '+str(Fewrror(),4)+' ) ')
Else
  MsgStop('Pasta '+cDelPath+' removida com sucesso.')
Endif
```

DISKSPACE()

Função de ambiente que determina quantos bytes estão disponíveis em uma determinada unidade de disco. Esta função obtém a informação sempre relativa à estação onde está sendo executado o Protheus Remote. Através do parâmetro nDrive , selecionamos qual a unidade de disco que desejamos obter a informação do espaço livre , onde:

- 0 : Unidade de disco atual da estação (DEFAULT).**
- 1 : Drive A: da estação remota.**
- 2 : Drive B: da estação remota.**
- 3 : Drive C: da estação remota.**
- 4 : Drive D: da estação remota ... e assim por diante.**

Caso a função DiskSpace seja executada através de um Job (processo isolado no Servidor , sem interface Remota) , ou seja passado um argumento de unidade de disco inexistente ou indisponível , a função DISKSPACE() retornará -1

☒ **Sintaxe:** DISKSPACE ([nDrive])

☒ **Parâmetros:**

nDrive	Número do drive, onde 0 é o espaço na unidade de disco corrente, e 1 é o drive A: do cliente, 2 é o drive B: do cliente, etc.
---------------	---

☒ **Retorno:**

Numérico	Número de bytes disponíveis no disco informado como parâmetro.
-----------------	--

Exemplo:

```
nBytesLocal := DISKSPACE( ) // Retorna o espaço disponível na unidade de disco local
IF nBytesLocal < 1048576
  MsgStop('Unidade de Disco local possui menos de 1 MB livre.')
Else
  MsgStop('Unidade de disco local possui '+str(nBytes_A,12)+' bytes livres.')
Endif
nBytes_A := DISKSPACE( 1 ) // Retorna o espaço disponível no drive A: local ( remote ).

If nBytes_A == -1
  MsgStop('Unidade A: não está disponível ou não há disco no Drive')
ElseIf nBytes_A < 8192
  MsgStop('Não há espaço disponível no disco. Substitua o disco na Unidade A:')
Else
  MsgStop('Unidade A: Verificada . '+str(nBytes_A,12)+' bytes livres.')
Endif
```

EXISTDIR()

Função utilizada para determinar se um path de diretório existe e é válido.

☑ **Sintaxe:** EXISTDIR (< cPath >)

☑ **Parâmetros:**

cPath	String contendo o diretório que será verificado, caso seja feita uma verificação a partir do server, devemos informar a partir do rootPath do Protheus, caso contrário devemos passar o path completo do diretório.
--------------	---

☑ **Retorno:**

Lógico	Retorna se verdadeiro(.T.) caso o diretório solicitado exista, falso(.F.) caso contrário.
---------------	---

Exemplo 01: No server a partir do rootPath

```
lRet := ExistDir('\teste')
```

Exemplo 02: No client, passando o FullPath

```
lRet := ExistDir('c:\APO')
```

FCLOSE()

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

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

☑ **Sintaxe:** FCLOSE (< nHandle >)

☑ **Parâmetros:**

nHandle	Handle do arquivo obtido previamente através de FOPEN() ou FCREATE().
----------------	---

☑ **Retorno:**

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

Exemplo:

```
#include "Fileio.ch"

nHandle := FCREATE("Testfile", FC_NORMAL)

If !FCLOSE(nHandle)
    conout( "Erro ao fechar arquivo, erro numero: ", FERROR() )
EndIf
```

FCREATE()

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

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

☒ **Atributos definidos no include FileIO.ch**

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

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

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

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

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

☒ **Retorno:**

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

FERASE()

Função utilizada para apagar um arquivo no disco . O Arquivo pode estar no Servidor ou na estação local (Remote). O arquivo para ser apagado deve estar fechado, não sendo permitido a utilização de caracteres coringa (wildcards).

☑ **Sintaxe:** FERASE (< cArquivo >)

☑ **Parâmetros:**

cArquivo	Nome do arquivo a ser apagado . Pode ser especificado um path absoluto ou relativo , para apagar arquivos na estação local (Remote) ou no Servidor, respectivamente.
-----------------	--

☑ **Retorno:**

Numérico	A função retornará 0 caso o arquivo seja apagado com sucesso , e -1 caso não seja possível apagar o arquivo. Caso a função retorne -1, é possível obter maiores detalhes da ocorrência através da função FERROR().
-----------------	--

Exemplo:

```
#include 'DIRECTORY.CH'

aEval(Directory("*.BAK"), { |aFile| FERASE(aFile[F_NAME]) })

// Este exemplo apaga um arquivo no cliente ( Remote ) , informando o status da operação
IF FERASE("C:\ListaTXT.tmp") == -1
    MsgStop('Falha na deleção do Arquivo ( FError'+str(ferror(),4)+ ' )')
Else
    MsgStop('Arquivo deletado com sucesso.')
ENDIF
```

FILE()

Função que verifica se existe um arquivo ou um padrão de arquivos, no diretório. Podem ser especificados caminhos absolutos (arquivos na estação - Remote) ou relativos (a partir do RootPath do Protheus Server) , sendo os caracteres "*" e "?" (wildcards) aceitos.

☑ **Sintaxe:** FILE (< cArquivo >)

☑ **Parâmetros:**

cArquivo	Nome do arquivo , podendo ser especificado um path (caminho) . Caminhos locais (Remote) ou caminhos de servidor são aceitos , bem como wildcards (Caracteres "*" e "?").
-----------------	--

☑ **Retorno:**

Lógico	O retorno será .T. caso o arquivo especificado exista. Caso o mesmo não exista no path especificado , a função retorna .F.
---------------	--

Exemplo:

```
//Verifica no diretório corrente do servidor se existe o arquivo teste.dbf
FILE("teste.dbf")

// Verifica no diretório Sigaadv do servidor se existe o arquivo teste.dbf
FILE("\SIGAADV\TESTE.dbf")

// Verifica no diretório Temp do cliente (Remote) se existe o arquivo teste.dbf
FILE("C:\TEMP\TESTE.dbf")
```

**Importante**

Caso a função FILE() seja executada em Job (programa sem interface remota), sendo passado um caminho absoluto de arquivo (exemplo c:\teste.txt) , a função retornará .F. e FERROR() retornará -1).

FILENOEXT()

Função que retorna o nome de um arquivo contido em uma string, ignorando a extensão.

☒ **Sintaxe:** FileNoExt(cString)

☒ **Parâmetros**

cString	String contendo o nome do arquivo.
----------------	------------------------------------

Exemplo:

```
Local cString := "\SIGAADV\ARQZZZ.DBF"
cString := FileNoExt( cString )
// Retorno → "\SIGAADV\ARQZZZ"
```

**Anotações**

FOPEN()

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

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

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



Importante

- FOPEN procura o arquivo no diretório corrente e nos diretórios configurados na variável de pesquisa do Sistema Operacional, a não ser que um path seja declarado explicitamente como parte do argumento <cArq>.
- Por serem executadas em um ambiente cliente-servidor, as funções de tratamento de arquivos podem trabalhar em arquivos localizados no cliente (estação) ou no servidor. O ADVPL identifica o local onde o arquivo será manipulado através da existência ou não da letra do drive no nome do arquivo passado em <cArq>. Ou seja, se o arquivo for especificado com a letra do drive, será aberto na estação. Caso contrário, será aberto no servidor com o diretório configurado como rootpath sendo o diretório raiz para localização do arquivo.

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

☑ **Parâmetros:**

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

☑ **Retorno:**

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

Exemplo:

```
#include 'fileio.ch'
...
nH := fopen('\sigaadv\error.log' , FO_READWRITE + FO_SHARED )
If nH == -1
    MsgStop('Erro de abertura : FERROR '+str(ferror(),4))
Else
    MsgStop('Arquivo aberto com sucesso.')
...
fclose(nH)
Endif
...
```

☒ **Tabela A: Modos de acesso a arquivos binários**

Modo	Constata(fileio.ch)	Operação
0	FO_READ	Aberto para leitura (padrão assumido)
1	FO_WRITE	Aberto para gravação
2	FO_READWRITE	Aberto para leitura e gravação

☒ **Tabela B: Modos de acesso de compartilhamento a arquivos binários**

Modo	Constata(fileio.ch)	Operação
0	FO_COMPAT	Modo de Compatibilidade (Default)
16	FO_EXCLUSIVE	Acesso total exclusivo
32	FO_DENYWRITE	Acesso bloqueando a gravação de outros processos ao arquivo.
48	FO_DENYREAD	Acesso bloqueando a leitura de outros processos ao arquivo.
64	FO_DENYNONE	Acesso compartilhado. Permite a leitura e gravação por outros.

**Anotações**

FREAD()

Função que realiza a leitura dos dados a partir um arquivo aberto, através de FOPEN(), FCREATE() e/ou FOPENPORT(), e armazena os dados lidos por referência no buffer informado. FREAD() lerá até o número de bytes informado em nQtyBytes; caso aconteça algum erro ou o arquivo chegue ao final, FREAD() retornará um número menor que o especificado em nQtyBytes. FREAD() lê normalmente caracteres de controle (ASC 128, ASC 0, etc.) e lê a partir da posição atual do ponteiro atual do arquivo, que pode ser ajustado ou modificado pelas funções FSEEK(), FWRITE() ou FREADSTR().

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

☑ **Sintaxe:** FREAD (< nHandle > , < cBuffer > , < nQtyBytes >)

☑ **Parâmetros:**

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

☑ **Retorno:**

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

FREADSTR ()

Função que realiza a leitura de caracteres de um arquivo binário. FREADSTR() lê de um arquivo aberto, através de FOPEN(), FCREATE(), FOPENPORT(). FREADSTR() lerá até o número de bytes informado em nQtyBytes ou até encontrar um CHR(0). Caso aconteça algum erro ou o arquivo chegue ao final, FREADSTR() retornará uma string menor do que nQtyBytes e colocará o erro em FERROR(). FREADSTR() lê a partir da posição atual do ponteiro, que pode ser ajustado pelo FSEEK(), FWRITE() ou FREAD().

☑ **Sintaxe:** FREADSTR (< nHandle > , < nQtyBytes >)

☑ **Parâmetros:**

nHandle	É o manipulador retornado pelas funções FOPEN(), FCREATE(), FOPENPORT().
nQtyBytes	Número máximo de bytes que devem ser lidos.

☑ **Retorno:**

Caracter	Retorna uma string contendo os caracteres lidos.
-----------------	--

FRENAME()

Através da função FRENAME() é possível renomear um arquivo para outro nome, tanto no servidor como na estação. Ao renomear um arquivo não esqueça que este arquivo deverá estar fechado (isto é , não pode estar em uso por nenhum outro processo ou estação). Caso o arquivo esteja aberto por outro processo , a operação de renomear o arquivo não é possível. A função fRename() não aceita wildcards (* e/ou ?).

Vale lembrar que não é possível renomear um arquivo especificando nos parâmetros simultaneamente um caminho de servidor e um de estação remota, bem como especificar dois arquivos remotos e executar a função fRename() através de um JOB. Caso isto ocorra, a função retornará -1 , e fError() retornará também -1.



Importante

Quando especificamos um path diferente nos arquivos de origem e destino , a função fRename() realiza a funcionalidade de MOVER o arquivo para o Path especificado.

☑ **Sintaxe:** FRENAME (< cOldFile > , < cNewFile >)

☑ **Parâmetros:**

cOldFile	Nome do arquivo será renomeado, aceita caminhos do servidor e caminhos do cliente. Caso não seja especificado nenhuma unidade de disco e path, é considerado o path atual no servidor.
cNewFile	Novo nome do arquivo, aceita também caminho do servidor, e caminho do cliente.

☑ **Retorno:**

Numérico	Se o status retornado for -1 , ocorreu algum erro na mudança de nome : Verifique se os dois caminhos estão no mesmo ambiente, verifique a existência do arquivo de origem, se ele não está em uso no momento por outro processo , e verifique se o nome do arquivo de destino já não existe no path de destino especificado.
-----------------	--



Anotações

FSEEK()

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

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

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

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

- ☑ **Parâmetros:**

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

- ☑ **Retorno:**

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

FT_FEOF()

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

- ☑ **Sintaxe: FT_FEOF ()**

- ☑ **Parâmetros:**

Nenhum	.
---------------	---

- ☑ **Retorno:**

Lógico	Retorna true caso o ponteiro do arquivo tenha chegado ao final, false caso contrário.
---------------	---

FT_FGOTO()

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

☑ **Sintaxe:** FT_FGOTO (< nPos >)

☑ **Parâmetros:**

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

☑ **Retorno:**

Nenhum	.
---------------	---

FT_FGOTOP()

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

☑ **Sintaxe:** FT_FGOTO (< nPos >)

☑ **Parâmetros:**

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

☑ **Retorno:**

Nenhum	.
---------------	---

FT_FLASTREC()

Função que retorna o número total de linhas do arquivo texto aberto pela FT_FUse. As linhas são delimitadas pela sequência de caracteres CRLF ou LF.



Dica

Verifique maiores informações sobre formato do arquivo e tamanho máximo da linha de texto na função FT_FREADLN().

☑ **Sintaxe:** FT_FLASTREC ()

☑ **Parâmetros:**

Nenhum	.
---------------	---

☑ **Retorno:**

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

Exemplo:

```

FT_FUse('teste.txt') // Abre o arquivo
CONOUT("Linhas no arquivo [" + str(ft_flastrec(),6) + "]\n")
FT_FGOTOP()
While !FT_FEOF()
  conout("Ponteiro [" + str(FT_FRECN(),6) + "]\n" Linha [" + FT_FREADLN() + "]\n")
  FT_FSKIP()
Enddo
FT_FUse() // Fecha o arquivo

```

FT_FREADLN()

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



- A utilização desta função não altera a posição do ponteiro para leitura dos dados, o ponteiro do arquivo não é movido. A movimentação do ponteiro é realizada através da função FT_FSKIP()
- O limite de 1022 bytes por linha inclui os caracteres delimitadores de final de linha. Deste modo, quando utilizados os separadores CRLF, isto nos deixa 1020 bytes de texto, e utilizando LF, 1021 bytes. A tentativa de leitura de arquivos com linhas de texto maiores do que os valores especificados acima resultará na leitura dos 1023 primeiros bytes da linha, e incorreta identificação das quebras de linha posteriores.
- As funções FT_F* foram projetadas para ler arquivos com conteúdo texto apenas. A utilização das mesmas em arquivos binários pode gerar comportamentos inesperados na movimentação do ponteiro de leitura do arquivo, e incorretas identificações nos separadores de final de linha.



- **Release:** Quando utilizado um Protheus Server, com build superior a 7.00.050713P, a função FT_FREADLN() também é capaz de ler arquivos texto / ASCII, que utilizam também o caractere LF ($\text{chr}(10)$) como separador de linha.

☒ **Sintaxe:** FT_FREADLN()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

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

FT_FRECNO()

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

A função FT_FRecno retorna a posição corrente do ponteiro do arquivo texto aberto pela FT_FUse.

☑ **Sintaxe:** FT_FRECNO ()

☑ **Parâmetros:**

Nenhum	.
--------	---

☑ **Retorno:**

Caracter	Retorna a posição corrente do ponteiro do arquivo texto.
----------	--

FT_FSKIP()

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

☑ **Sintaxe:** FT_FSKIP ([nLinhas])

☑ **Parâmetros:**

nLinhas	nLinhas corresponde ao número de linhas do arquivo TXT ref. movimentação do ponteiro de leitura do arquivo.
---------	---

☑ **Retorno**

Nenhum	.
--------	---

FT_FUSE()

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



Dica

Verifique maiores informações sobre formato do arquivo e tamanho máximo da linha de texto na função FT_FREADLN().

☑ **Sintaxe:** FT_FUSE ([cTXTFile])

☑ **Parâmetros:**

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

☑ **Retorno:**

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

FWRITE()

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

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

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

☑ **Parâmetros:**

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

☑ **Retorno:**

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

Exemplo:

```
#INCLUDE "FILEIO.CH"
#define F_BLOCK 1024 // Define o bloco de Bytes a serem lidos / gravados por vez

User Function TestCopy()
Local cBuffer := SPACE(F_BLOCK)
Local nHOrigem , nHDestino
Local nBytesLidos , nBytesFalta , nTamArquivo
Local nBytesLer , nBytesSalvo
Local lCopiaOk := .T.

// Abre o arquivo de Origem
nHOrigem := FOPEN("ORIGEM.TXT", FO_READ)
```

Exemplo (continuação):

```
// Testa a abertura do Arquivo
If nHOrigem == -1
    MsgStop('Erro ao abrir origem. Ferror = '+str(ferror(),4),'Erro')
    Return .F.
Endif

// Determina o tamanho do arquivo de origem
nTamArquivo := Fseek(nHOrigem,0,2)

// Move o ponteiro do arquivo de origem para o inicio do arquivo
Fseek(nHOrigem,0)

// Cria o arquivo de destino
nHDestino := FCREATE("DESTINO.TXT", FC_NORMAL)

// Testa a criação do arquivo de destino
If nHDestino == -1
    MsgStop('Erro ao criar destino. Ferror = '+str(ferror(),4),'Erro')
    FCLOSE(nHOrigem) // Fecha o arquivo de Origem
    Return .F.
Endif

// Define que a quantidade que falta copiar é o próprio tamanho do Arquivo
nBytesFalta := nTamArquivo

// Enquanto houver dados a serem copiados
While nBytesFalta > 0

    // Determina quantidade de dados a serem lidos
    nBytesLer := Min(nBytesFalta , F_BLOCK )

    // lê os dados do Arquivo
    nBytesLidos := FREAD(nHOrigem, @cBuffer, nBytesLer )

    // Determina se não houve falha na leitura
    If nBytesLidos < nBytesLer
        MsgStop( "Erro de Leitura da Origem. "+
                Str(nBytesLer,8,2)+" bytes a LER."+
                Str(nBytesLidos,8,2)+" bytes Lidos."+
                "Ferror = "+str(ferror(),4),'Erro')

        lCopiaOk := .F.
        Exit
    Endif

    // Salva os dados lidos no arquivo de destino
    nBytesSalvo := FWRITE(nHDestino, cBuffer,nBytesLer)

    // Determina se não houve falha na gravação
    If nBytesSalvo < nBytesLer
        MsgStop("Erro de gravação do Destino. "+
                Str(nBytesLer,8,2)+" bytes a SALVAR."+
                Str(nBytesSalvo,8,2)+" bytes gravados."+
                "Ferror = "+str(ferror(),4),'Erro')

        lCopiaOk := .F.
        EXIT
    Endif
```


Exemplo (continuação):

```
// Elimina do Total do Arquivo a quantidade de bytes copiados
nBytesFalta -= nBytesLer

Enddo

// Fecha os arquivos de origem e destino
FCLOSE(nHOrigem)
FCLOSE(nHDestino)

If lCopiaOk
    MsgStop('Cópia de Arquivos finalizada com sucesso. '+'
            str(nTamArquivo,12,0)+' bytes copiados.','Final')
Else
    MsgStop('Falha na Cópia. Arquivo de Destino incompleto. '+'
            'Do total de '+str(nTamArquivo,12,0)+' bytes, faltaram
            '+str(nBytesFalta,12,0)+' bytes.','Final')
Endif

Return
```

MSCOPYFILE()

Função que executa a cópia binária de um arquivo para o destino especificado.

☒ **Sintaxe:** MSCOPYFILE(cArqOrig, cArqDest)

☒ **Parâmetros:**

cArqOrig	Nome do arquivo origem e a extensão.
cArqDest	Nome do arquivo destino e a extensão.

☒ **Retorno:**

Lógico	Se a copia for realizada com sucesso a função retornará verdadeiro (.T.).
---------------	---

Exemplo:

```
Local cArqOrig := 'ARQ00001.DBF'
Local cArqDest := 'ARQ00002.XXX'

If MsCopyFile( cArqOrig, cArqDest )
    APMsgInfo('Copia realizada com sucesso!')
EndIf
```

MSCOPYTO()

Função que realiza a cópia dos registros de uma base de dados para outra, criando o arquivo destino de acordo com a estrutura da base de dados origem.

☑ **Sintaxe:** MSCOPYTO([cArqOrig], cArqDest)

☑ **Parâmetros:**

cArqOrig	Nome do arquivo origem e a extensão se o ambiente for Top o parâmetro passará a ser obrigatório.
cArqDest	Nome do arquivo destino e a extensão.

☑ **Retorno:**

Lógico	Se a copia for realizada com sucesso a função retornará verdadeiro (.T.).
---------------	---

Exemplo:

```
Local cArqDest := 'SX2ZZZ.DBF'  
DbSelectArea('SX2')  
If MsCopyTo( , cArqDest )  
    APMsgInfo('Copia realizada com sucesso!')  
Else  
    APMsgInfo('Problemas ao copiar o arquivo SX2!')  
EndIf
```

MSCREATE()

Função que cria um arquivo (tabela) de acordo com a estrutura informada no parâmetro aStruct. Se o parâmetro cDriver não for informado o RDD corrente será assumido como padrão. Para criação de tabelas no TopConnect é necessário estar conectado ao banco e o environment do protheus ser TOP.



Importante

aStruct: array contendo a estrutura da tabela aonde:
1º - caracter, nome do campo;
2º - caracter, tipo do campo;
3º - numérico, tamanho do campo;
4º - numérico, decimais.

☑ **Sintaxe:** MsCreate(cArquivo, aStru ,[cDriver])

☑ **Parâmetros:**

cArquivo	Nome do arquivo.
aStruct	Estrutura do arquivo.
cDriver	RDD do arquivo.

☒ **Retorno:**

Lógico	Indica se a operação foi executada com sucesso.
---------------	---

Exemplo:

```
Local cTarget := '\sigaadv\  
Local aStrut  
aStrut := { { 'Campo', 'C', 40, 0 } }  
If MsCreate( cTarget+'ARQ1001', aStrut )  
    APMsInfo('Criado com sucesso!')  
Else  
    APMsInfo('Problemas ao criar o arquivo!')  
EndIf
```

MSERASE()

Função utilizada para deletar fisicamente o arquivo especificado.

☒ **Sintaxe:** MsErase(cArquivo, [cIndice], [cDriver])

☒ **Parâmetros:**

cArquivo	Nome do arquivo e a extensão.
cIndice	Nome do arquivo de índice e a extensão.
cDriver	RDD do arquivo, se não for informado assumirá o RDD corrente.

☒ **Retorno:**

Lógico	Indica se a operação foi executada com sucesso.
---------------	---

Exemplo:

```
Local cArquivo := 'SX2ZZZ.DBF'  
Local cIndice := 'SX2ZZZ'+ OrdBagExt()  
If MsErase( cArquivo, cIndice )  
    APMsInfo( 'Arquivo deletado com sucesso!' )  
Else  
    APMsInfo( 'Problemas ao deletar arquivo!' )  
EndIf
```



Anotações

MSRENAME()

Função que verifica a existência do arquivo especificado.

☑ **Sintaxe:** `MsFile(cArquivo, [cIndice], [cDriver])`

☑ **Parâmetros:**

cArquivo	Nome do arquivo e a extensão.
cIndice	Nome do arquivo de índice e a extensão.
cDriver	RDD do arquivo, se não for informado assumirá o RDD corrente.

☑ **Retorno:**

Lógico	Indica se o arquivo especificado existe.
---------------	--

Exemplo:

```
Local cArquivo := 'SX2ZZZ.DBF'  
Local cIndice := 'SX2ZZZ'+ OrdBagExt()  
If !MsFile ( cArquivo, cIndice )  
    APMsgInfo( 'Arquivo não encontrado!' )  
EndIf
```

RETFILENAME()

Função que retorna o nome de um arquivo contido em uma string, ignorando o caminho e a extensão.

☑ **Sintaxe:** `RetFileName(cArquivo)`

☑ **Parâmetros:**

cArquivo	String contendo o nome do arquivo
-----------------	-----------------------------------

☑ **Retorno:**

Caracter	Nome do arquivo contido na string cArquivo sem o caminho e a extensão.
-----------------	--

Exemplo:

```
Local cArquivo := '\SIGAADV\ARQZZZ.DBF'  
cArquivo := RetFileName( cArquivo )  
// retorno 'ARQZZZ'
```

Manipulação de arquivos e índices temporários

CRIATRAB()

Função que cria um arquivo de trabalho com uma estrutura especificada, sendo que:

- ☐ Caso o parâmetro IDbf seja definido como .T., a função criará um arquivo DBF com este nome e a estrutura definida em aArray.
- ☐ Caso o parâmetro IDbf seja definido como .F., a função não criará arquivo de nenhum tipo, apenas fornecerá um nome válido.

☑ **Sintaxe:** CriaTrab(aArray,IDbf)

☑ **Parâmetros:**

aArray	Array multidimensional contendo a estrutura de campos da tabela que será criada no formato: {Nome, Tipo, Tamanho, Decimal}
IDbf	Determina se o arquivo de trabalho deve ser criado (.T.) ou não (.F.)

☑ **Retorno:**

Caracter	Nome do Arquivo gerado pela função.
-----------------	-------------------------------------

Exemplo:

```
// Com IDbf = .F.  
cArq := CriaTrab(NIL, .F.)  
cIndice := "C9_AGREG"+IndexKey()  
Index on &cIndice To &cArq  
  
// Com IDbf = .T.  
aStru := {}  
AADD(aStru,{ "MARK", "C", 1, 0})  
AADD(aStru,{ "AGLUT", "C", 10, 0})  
AADD(aStru,{ "NUMOP", "C", 10, 0})  
AADD(aStru,{ "PRODUTO", "C", 15, 0})  
AADD(aStru,{ "QUANT", "N", 16, 4})  
AADD(aStru,{ "ENTREGA", "D", 8, 0})  
AADD(aStru,{ "ENTRAJU", "D", 8, 0})  
AADD(aStru,{ "ORDEM", "N", 4, 0})  
AADD(aStru,{ "GERADO", "C", 1, 0})  
cArqTrab := CriaTrab(aStru, .T.)  
USE &cArqTrab ALIAS TRB NEW
```



Importante

Na criação de índices de trabalho temporários é utilizada a sintaxe:

☐ **CriaTrab(Nil, .F.)**

Manipulação de bases de dados

ALIAS()

Função de banco de dados utilizada para determinar o alias da área de trabalho especificada. Alias é o nome atribuído a uma área de trabalho quando um arquivo de banco de dados está em uso. O nome real atribuído é o nome do arquivo de banco de dados, ou um nome que foi explicitamente atribuído através da cláusula ALIAS do comando USE.

A função ALIAS() é o inverso da função SELECT() pois retorna o alias através do número da área de trabalho, enquanto SELECT() retorna o número da área de trabalho através do alias.

☑ **Sintaxe:** ALIAS ([nAreaTrabalho])

☑ **Parâmetros:**

nAreaTrabalho	<nAreaTrabalho> é o número da área de trabalho a ser verificada.
----------------------	--

☑ **Retorno:**

Caracter	Retorna o alias da área de trabalho especificada na forma de uma cadeia de caracteres, em letra maiúscula. Caso <nAreaTrabalho> não seja especificada, é retornado o alias da área de trabalho corrente. Se não houver nenhum arquivo de banco de dados em uso na área de trabalho especificada, ALIAS() retorna uma cadeia de caracteres nula ("").
-----------------	--

Exemplo:

```
cAlias := alias()
IF empty(cAlias)
    alert('Não há Area em uso')
Else
    alert('Area em uso atual : '+cAlias)
Endif
```

BOF() / EOF()

As funções BOF() e EOF() são utilizadas para determinar se o ponteiro de leitura do arquivo encontra-se no começo ou no final do mesmo conforme abaixo:

- ☐ BOF() é uma função de tratamento de banco de dados utilizada para testar uma condição de limite de inicial do arquivo quando o ponteiro de registros está se movendo para trás em um arquivo de banco de dados.
- ☐ EOF() é uma função de tratamento de banco de dados utilizada para testar uma condição de limite de final de arquivo quando o ponteiro de registros está se movendo para frente em um arquivo de banco de dados.

Normalmente é utilizada a condição EOF() como parte do argumento <ICondicao> de uma construção DO WHILE que processa registros sequencialmente em um arquivo de banco de dados. Neste caso <ICondicao> incluiria um teste para .NOT. EOF(), forçando o laço DO WHILE a terminar quando EOF() retornar verdadeiro (.T.)

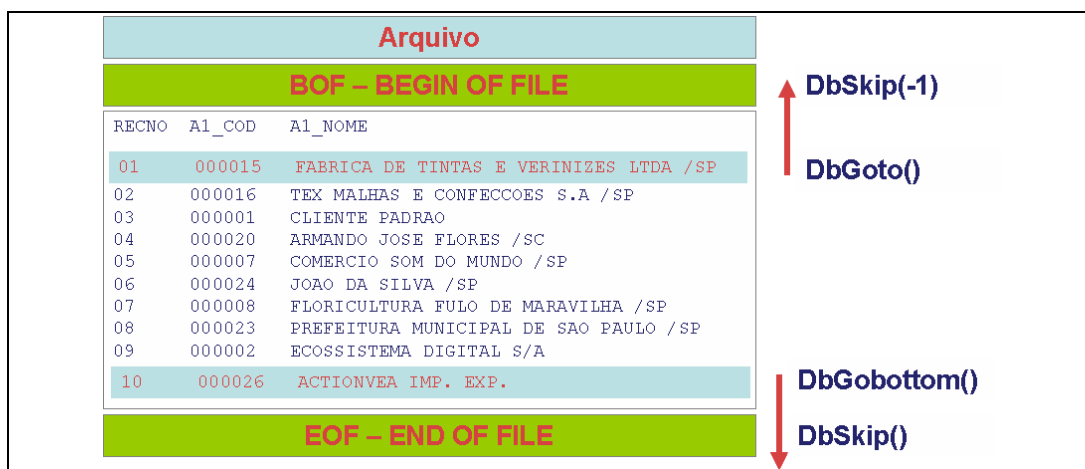
☑ **Sintaxe:** BOF() / EOF()

☑ **Parâmetros:**

Nenhum	.
--------	---

☑ **Retorno:**

Lógico	Retorna verdadeiro (.T.) quando , feita uma tentativa de mover o ponteiro de registros para além do primeiro registro lógico em um arquivo de banco de dados, do contrário, ela retorna falso (.F.).
Lógico	Retorna verdadeiro (.T.) quando , feita uma tentativa de mover o ponteiro de registros para além do último registro lógico em um arquivo de banco de dados, do contrário, ela retorna falso (.F.). Caso nao haja nenhum arquivo de banco de dados aberto na área de trabalho corrente, EOF() retorna falso (.F.). Se o arquivo de banco de dados corrente não possui registros, EOF() retorna verdadeiro (.T.).



COPY()

O comando COPY TO permite a cópia de todos ou parte dos registros da tabela atualmente selecionada como área de trabalho atual, para um novo arquivo. Os registros considerados para a cópia podem ser limitados pela cláusula <escopo>, através de expressões FOR/WHILE, e/ou através de um filtro.

Se o filtro para registros deletados (SET DELETED) estiver desligado (OFF), registros deletados (marcados para deleção) são copiados para o arquivo de destino, mantendo este status. Caso contrário, nenhum registro deletado é copiado. Da mesma maneira, caso exista uma condição de filtro na tabela atual (SET FILTER), apenas os registros que satisfaçam a condição de filtro serão copiados.

Os registros são lidos na tabela atual, respeitando a ordem de índice setada. Caso não haja índices abertos, ou a ordem de navegação nos índices (SET ORDER) seja 0 (zero), os registros são lidos em ordem natural (ordem de RECNO) .

A tabela de destino dos dados copiados é criada, e aberta em modo exclusivo, antes da operação de cópia efetiva ser iniciada.

☒ **Tabela A : Especificação do formato SDF (System Data Format)**

Elemento do Arquivo	Formato
Campos 'C' Caractere	Tamanho fixo, ajustado com espaços em branco
Campos 'D' Data	Formato aaaammdd (ano, mês, dia)
Campos 'L' lógicos	T ou F
Campos 'M' Memo	(campo ignorado)
Campos 'N' Numéricos	Ajustados à direita, com espaços em branco.
Delimitador de Campos	Nenhum
Separador de Registros	CRLF (ASCII 13 + ASCII 10)
Marca de final de arquivo (EOF)	Nenhum

☒ **Tabela B : Especificação do formato delimitado (DELIMITED / DELIMITED WITH <cDelimiter>)**

Elemento do Arquivo	Formato
Campos 'C' Caractere	Delimitados, ignorando espaços à direita
Campos 'D' Data	Formato aaaammdd (ano, mês, dia)
Campos 'L' lógicos	T ou F
Campos 'M' Memo	(campo ignorado)
Campos 'N' Numéricos	sem espaços em branco.
Delimitador de Campos	Vírgula
Separador de Registros	CRLF (ASCII 13 + ASCII 10)
Marca de final de arquivo (EOF)	Nenhum



Importante

A Linguagem Advpl, antes do Protheus, suportava a geração de uma tabela delimitada diferenciada, obtida através do comando *COPY TO (...) DELIMITED WITH BLANK* . No Protheus este formato não é suportado. Caso utilize-se este comando com a sintaxe acima, o arquivo ASCII gerado será delimitado, utilizando-se a sequência de caracteres 'BLANK' como delimitadora de campos Caractere.

☒ **Sintaxe:**

```
COPY [ FIELDS <campo,...> ] TO cFile [cEscopo] [ WHILE <ICondicao> ]
[ FOR <ICondicao> ] [ SDF | DELIMITED [WITH <cDelimiter>] ]
[ VIA <cDriver> ]
```


☒ **Parâmetros:**

FIELDS <campo,...>	FIELDS <campo,...> especifica um ou mais campos, separados por vírgula, a serem copiados para a tabela de destino. Caso não especificado este parâmetro, serão copiados todos os campos da tabela de origem.
TO cFile	TO <cFile> especifica o nome do arquivo de destino. O nome do arquivo de destino pode ser especificado de forma literal direta, ou como uma expressão Advpl, entre parênteses. Caso sejam especificadas as cláusulas SDF ou DELIMITED, é gerado um arquivo ASCII, com extensão .txt por default.
cEscopo	<cEscopo> define a porção de dados da tabela atual a ser copiada. Por default, são copiados todos os registros (ALL). Os escopos possíveis de uso são: ALL - Copia todos os registros. REST - Copia, a partir do registro atualmente posicionado, até o final da tabela. NEXT <n> - Copia apenas <n> registros, iniciando a partir do registro atualmente posicionado. OBSERVAÇÃO : Vale a pena lembrar que o escopo é sensível também às demais condições de filtro (WHILE / FOR).
WHILE <ICondicao>	WHILE <ICondicao> permite especificar uma condição para realização da cópia, a partir do registro atual, executada antes de inserir cada registro na tabela de destino, sendo realizada a operação de cópia enquanto esta condição for verdadeira.
FOR <ICondicao>	FOR <ICondicao> especifica uma condição para cópia de registros, executada antes de inserir um registro na tabela de destino, sendo a operação realizada apenas se ICondicao ser verdadeira (.T.)
[SDF DELIMITED]	[SDF DELIMITED [WITH <xcDelimiter>]] SDF especifica que o tipo de arquivo de destino gerado é um arquivo no formato "System Data Format" ASCII, onde registros e campos possuem tamanho fixo no arquivo de destino. DELIMITED especifica que o arquivo ASCII de destino será no formato delimitado, onde os campos do tipo Caractere são delimitados entre aspas duplas (delimitador Default). Registros e campos têm tamanho variável no arquivo ASCII. DELIMITED WITH <xcDelimiter> permite especificar um novo caractere, ou sequência de caracteres, a ser utilizada como delimitador, ao invés do default (aspas duplas). O caractere delimitador pode ser escrito de forma literal, ou como uma expressão entre parênteses. Nas Tabelas complementares A e B, na documentação do comando, são detalhadas as especificações dos formatos SDF e DELIMITED.
VIA <cDriver>	VIA <xcDriver> permite especificar o driver utilizado para criar a tabela de destino dos dados a serem copiados. O Driver deve ser especificado como uma expressão caractere. Caso especificado como um valor literal direto, o mesmo deve estar entre aspas.

☒ **Retorno:**

Nenhum	.
---------------	---

COPY STRUCTURE()

O comando COPY STRUCTURE cria uma nova tabela vazia, com a estrutura da tabela ativa na área de trabalho atual. Se a tabela a ser criada já exista, a mesma é sobrescrita. A tabela de destino criada utiliza o mesmo RDD da tabela de origem (tabela ativa na área de trabalho atual).



Importante

A Linguagem Advpl, antes do Protheus, suportava a parametrização de uma lista de campos da tabela atual, para compor a estrutura da tabela de destino, através da cláusula *FIELDS* <campo,...>. Esta opção não é suportada no Protheus. Caso seja utilizada, o programa será abortado com a ocorrência de erro fatal : **'DBCOPYStruct - Parameter <Fields> not supported in Protheus'**

☒ Sintaxe:

COPY STRUCTURE TO <xcDataBase>

☒ Parâmetros:

TO <xcDataBase>

Deve ser especificado em xcDatabase o nome da tabela a ser criada.

☒ Retorno:

Nenhum

.

DBAPPEND()

A função DBAPPEND() acrescenta mais um registro em branco no final da tabela corrente. Se não houver erro da RDD, o registro é acrescentado e bloqueado.

☒ Sintaxe: DBAPPEND ([ILiberaBloqueios])

☒ Parâmetros:

ILiberaBloqueios

Se o valor for .T., libera todos os registros bloqueados anteriormente (locks). Se for .F., todos os bloqueios anteriores são mantidos. Valor default: .T.

☒ Retorno:

Nenhum

.

Exemplo:

```
USE Clientes NEW
FOR i:=1 to 5
  DBAPPEND(.F.)
  NOME := "XXX"
  END := "YYY"
NEXT
// Os 5 registros incluídos permanecem bloqueados
DBAPPEND()
// Todos os bloqueios anteriores são liberados
```

DBCLEARALLFILTER()

A função DBCLEARALLFILTER() salva as atualizações realizadas e pendentes de todas as tabelas e depois limpa as condições de filtro de todas as tabelas.

☒ **Sintaxe:** DBCLEARALLFILTER()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Nenhum	.
--------	---

Exemplo:

```
USE Clientes NEW
DBSETFILTER( {|| Idade < 40}, 'Idade < 40') // Seta a expressão de filtro
...
DBCLEARALLFILTER()
// Limpa a expressão de filtro de todas as ordens
```



Anotações

DBCLEARFILTER()

A função DBCLEARFILTER() salva as atualizações realizadas e pendentes na tabela corrente e depois limpa todas as condições de filtro da ordem ativa no momento. Seu funcionamento é oposto ao comando SET FILTER.

☒ **Sintaxe:** DBCLEARFILTER()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Nenhum	.
--------	---

Exemplo:

```
USE Clientes NEW
DBSETFILTER( {|| Idade < 40}, "Idade < 40" ) // Seta a expressão de filtro
...
DBCLEARFILTER()
// Limpa a expressão de filtro
```

DBCLEARINDEX()

A função DBCLEARINDEX() salva as atualizações pendentes na tabela corrente e fecha todos os arquivos de índice da área de trabalho. Por consequência, limpa todas as ordens da lista. Seu funcionamento é oposto ao comando SET INDEX.

☒ **Sintaxe:** DBCLEARINDEX()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Nenhum	.
--------	---

Exemplo:

```
USE Clientes NEW
DBSETINDEX("Nome") // Abre o arquivo de índice "Nome"
...
DBCLEARINDEX()
// Fecha todos os arquivos de índices
```

DBCLOSEALL()

A função DBCLOSEALL() salva as atualizações pendentes, libera todos os registros bloqueados e fecha todas as tabelas abertas (áreas de trabalho) como se chamasse DBCLOSEAREA para cada área de trabalho.

☒ **Sintaxe:** DBCLOSEALL()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Nenhum	.
--------	---

Exemplo:

```
// Este exemplo demonstra como se pode utilizar o DBCLOSEALL para fechar a área de
trabalho atual.
USE Clientes NEW
DBSETINDEX("Nome") // Abre o arquivo de índice "Nome"
USE Fornecedores NEW
DBSETINDEX("Idade") // Abre o arquivo de índice "Idade"
...
DBCLOSEALL() //Fecha todas as áreas de trabalho, todos os índices e ordens
```

DBCLOSEAREA()

A função DBCLOSEAREA() permite que um alias presente na conexão seja fechado, o que viabiliza seu reuso em outra operação. Este comando tem efeito apenas no alias ativo na conexão, sendo necessária sua utilização em conjunto com o comando DbSelectArea().

☒ **Sintaxe:** DBCLOSEAREA()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Nenhum	.
--------	---

Exemplo:

```
DbUserArea(.T., "DBFCDX", "\SA1010.DBF", "SA1DBF", .T., .F.)
DbSelectArea("SA1DBF")
MsgInfo("A tabela SA1010.DBF possui:" + STRZERO(RecCount(),6) + " registros.")
DbCloseArea()
```

DBCOMMIT()

A função DBCOMMIT() salva em disco todas as atualizações pendentes na área de trabalho corrente.

☒ **Sintaxe:** DBCOMMIT()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Nenhum	.
--------	---

Exemplo:

```
USE Clientes NEW
DBGOTO(100)
Nome := "Jose"
USE Fornecedores NEW
DBGOTO(168)
Nome := "Joao"
DBCOMMIT() // Salva em disco apenas as alterações realizadas na tabela Fornecedores
```

DBCOMMITALL()

A função DBCOMMITALL() salva em disco todas as atualizações pendentes em todas as áreas de trabalho.

☒ **Sintaxe:** DBCOMMITALL()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Nenhum	.
--------	---

Exemplo:

```
USE Clientes NEW
DBGOTO(100)
Nome := "Jose"
USE Fornecedores NEW
DBGOTO(168)
Nome := "Joao"
DBCOMMITALL()
// Salva em disco as alterações realizadas nas tabelas Clientes e Fornecedores
```

DBCCREATE()

A função DBCREATE() é utilizada para criar um novo arquivo de tabela cujo nome está especificado através do primeiro parâmetro (cNome) e estrutura através do segundo (aEstrutura). A estrutura é especificada através de um array com todos os campos, onde cada campo é expresso através de um array contendo {Nome, Tipo, Tamanho, Decimais}.

☑ **Sintaxe:** DBCREATE (< cNOME > , < aESTRUTURA > , [cDRIVER])

☑ **Parâmetros:**

cNOME	Nome do arquivo a ser criado. Se contém pasta, ela se localiza abaixo do "RootPath". Se não, é criado por padrão no caminho formado por "RootPath"+"StartPath"
aESTRUTURA	Lista com as informações dos campos para ser criada a tabela.
cDRIVER	Nome da RDD a ser utilizado para a criação da tabela. Se for omitido será criada com a corrente.

☑ **Retorno:**

Nenhum	.
---------------	---

Exemplo:

```
// Este exemplo mostra como se pode criar novo arquivo através da função DBCREATE:
LOCAL aEstrutura :={{Cod,N,3,0},
                    {Nome,C,10,0},
                    {Idade,N,3,0},
                    {Nasc,D,8,0},
                    {Pagto,N,7,2}}
// Cria a tabela com o RDD corrente
DBCREATE('\teste\cliente.dbf', aEstrutura)
USE '\teste\cliente.dbf' VIA 'DBFCDX' NEW
```



Importante

Erros mais comuns:

1. *DBCcreate - Data base files can only be created on the server:* O nome do arquivo a ser criado não pode conter 'driver', pois, por convenção, ele seria criado na máquina onde o Remote está rodando.
2. *DBCcreate - Invalid empty filename:* Nome do arquivo não foi especificado
3. *DBCcreate - Field's name cannot be 'DATA':* Algumas RDD's não suportam este nome de campo. É uma palavra reservada.
4. *DBCcreate - The length of Field's name must be at most 10:* Nome do campo não pode ter mais que 10 caracteres.
5. *DBCcreate - Field's name must be defined:* Nome do campo não foi definido.
6. *DBCcreate - Field's type is not defined:* Tipo do campo não foi definido.
7. *DBCcreate - invalid Field's type:* Tipo do campo é diferente de 'C', 'N', 'D', 'M' ou 'L'.
8. *DBCcreate - Invalid numeric field format:* Considerando 'len' o tamanho

total do campo numérico e 'dec' o número de decimais, ocorre este erro se:

- (len = 1) .and. (dec <> 0): Se o tamanho total é 1, o campo não pode ter decimais
- (len>1) .and. (len< dec + 2): Se o tamanho total é maior que 1, ele deve ser maior que o número de decimais mais 2, para comportar o separador de decimais e ter pelo menos um algarismo na parte inteira.

Exemplo: O número 12.45 poderia ser o valor de um campo com len=5 e dec=2 (no mínimo).



Erros mais comuns:

Podem ocorrer também erros decorrentes de permissão e direitos na pasta onde se está tentando criar o arquivo ou por algum problema no banco de dados. Verifique as mensagens do servidor Protheus e do banco de dados.

DBCREATEINDEX()

A função DBCREATEINDEX() é utilizada para criar um novo arquivo de índice com o nome especificado através do primeiro parâmetro, sendo que se o mesmo existir é deletado e criado o novo. Para tanto são executados os passos a seguir:

- ☐ Salva fisicamente as alterações ocorridas na tabela corrente;
- ☐ Fecha todos os arquivos de índice abertos;
- ☐ Cria o novo índice;
- ☐ Seta o novo índice como a ordem corrente;
- ☐ Posiciona a tabela corrente no primeiro registro do índice.

Com exceção do RDD CTREE, a tabela corrente não precisa estar aberta em modo exclusivo para a criação de índice, pois na criação de índices no Ctree é alterada a estrutura da tabela, precisando para isto a tabela estar aberta em modo exclusivo.

☑ **Sintaxe:** **DBCREATEINDEX(<cNOME>, <cEXPCHAVE>, [bEXPCHAVE], [IUNICO])**

☑ **Parâmetros:**

cNOME	Nome do arquivo de índice a ser criado.
cEXPCHAVE	Expressão das chaves do índice a ser criado na forma de string.
bEXPCHAVE	Expressão das chaves do índice a ser criado na forma executável.
IUNICO	Cria índice como único (o padrão é .F.).

☑ **Retorno:**

Nenhum	.
---------------	---

Exemplo:

```
// Este exemplo mostra como se pode criar novo arquivo de índice criando a ordem sobre os  
// campos Nome e End e não aceitará duplicação:
```

```
USE Cliente VIA "DBFCDX" NEW  
DBCINDEX("teste\ind2.cdx", "Nome+End", { || Nome+End }, .T.)
```

DBDELETE()

A função DBDELETE() marca o registro corrente como "apagado" logicamente(), sendo necessária sua utilização em conjunto com as funções RecLock() e MsUnlock().

Para filtrar os registro marcados do alias corrente pode-se utilizar o comando SET DELETED e para apagá-los fisicamente pode-se utilizar a função __DBPACK().

☒ **Sintaxe:** DBDELETE ()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Nenhum	.
--------	---

Exemplo:

```
DbSelectArea("SA1")  
DbSetOrder(1) // A1_FILIAL + A1_COD + A1_LOJA  
DbSeek("01" + "900001" + "01") // Busca exata
```

```
IF Found()  
RecLock("SA1", .F.) // Define que será realizada uma alteração no registro posicionado  
DbDelete() // Efetua a exclusão lógica do registro posicionado.  
MsUnlock() // Confirma e finaliza a operação  
ENDIF
```



Anotações

DBF()

A função DBF() verifica qual é o Alias da área de trabalho corrente. O Alias é definido quando a tabela é aberta através do parâmetro correspondente (DBUSEAREA()). Esta função é o inverso da função SELECT(), pois nesta é retornado o número da área de trabalho do Alias correspondente.

☒ **Sintaxe:** DBF()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Caracter	Retorna o Alias corrente. Caso não exista Alias corrente retorna "" (String vazia).
----------	---

Exemplo:

```
dbUseArea( .T., "dbfcdxads", "\\dadosadv609\sa1990.dbf", "SSS", .T., .F. )  
MessageBox("O Alias corrente é: "+DBF(), "Alias", 0) //Resultado: "O Alias corrente é: SSS"
```

DBFIELDINFO()

A função DBFIELDINFO() é utilizada para obter informações sobre determinado campo da tabela corrente. O tipo de informação (primeiro argumento) é escolhido de acordo com as constantes abaixo:

☒ **Tabela A : Constantes utilizadas na parametrização da função**

Constante	Descrição	Retorno
DBS_NAME	Nome do campo.	Caracter
DBS_DEC	Número de casas decimais.	Numérico
DBS_LEN	Tamanho.	Numérico
DBS_TYPE	Tipo.	Caracter

A posição do campo não leva em consideração os campos internos do Protheus (Recno e Deleted).

☒ **Sintaxe:** DBFIELDINFO (< nINFOTIPO > , < nCAMPO >)

☒ **Parâmetros:**

nINFOTIPO	Tipo de informação a ser verificada (DBS_NAME, DBS_DEC, DBS_LEN e DBS_TYPE).
nCAMPO	Posição do campo a ser verificado.

☒ **Retorno:**

Indefinido	Retorna NIL se não há tabela corrente ou a posição do campo especificado está inválida. Informação do campo Informação requisitada pelo usuário (pode ser de tipo numérico se for tamanho ou casas decimais, tipo caracter se for nome ou tipo).
-------------------	---

Exemplo:

USE Clientes NEW

DBFIELDINFO(DBS_NAME,1) // Retorno: Nome

DBFIELDINFO(DBS_TYPE,1) // Retorno: C

DBFIELDINFO(DBS_LEN,1) // Retorno: 10

DBFIELDINFO(DBS_DEC,1) // Retorno: 0

DBFILTER()

A função DBFILTER() é utilizada para verificar a expressão de filtro ativo na área de trabalho corrente.

☒ **Sintaxe:** DBFILTER()

☒ **Parâmetros:**

Nenhum	.
---------------	---

☒ **Retorno:**

Caracter	Retorna a expressão do filtro ativo na área de trabalho atual. Caso não exista filtro ativo retorna "" (String vazia).
-----------------	--

Exemplo:

USE Cliente INDEX Ind1 NEW

SET FILTER TO Nome > "Jose"

DBFILTER() // retorna: Nome > "Jose"

SET FILTER TO Num < 1000

DBFILTER() // retorna: Num < 1000



Anotações

DBGOTO()

Move o cursor da área de trabalho ativa para o record number (recno) especificado, realizando um posicionamento direto, sem a necessidade uma busca (seek) prévio.

- ☒ **Sintaxe: DbGoto(nRecno)**
- ☒ **Parâmetros**

nRecno	Record number do registro a ser posicionado.
---------------	--

Exemplo:

```
DbSelectArea("SA1")
DbGoto(100) // Posiciona no registro 100

IF !EOF() // Se a área de trabalho não estiver em final de arquivo
    MsgInfo("Você está no cliente:" + A1_NOME)
ENDIF
```

DBGOTOP()

Move o cursor da área de trabalho ativa para o primeiro registro lógico.

- ☒ **Sintaxe: DbGoTop()**
- ☒ **Parâmetros**

Nenhum	.
---------------	---

Exemplo:

```
nCount := 0 // Variável para verificar quantos registros há no intervalo
DbSelectArea("SA1")
DbSetOrder(1) // A1_FILIAL + A1_COD + A1_LOJA
DbGoTop()

While !BOF() // Enquanto não for o início do arquivo
    nCount++ // Incrementa a variável de controle de registros no intervalo
    DbSkip(-1)
End

MsgInfo("Existem :"+STRZERO(nCount,6)+ "registros no intervalo").

// Retorno esperado :000001, pois o DbGoTop posiciona no primeiro registro.
```

DBGOBOTTON()

Movimenta o cursor da área de trabalho ativa para o último registro lógico.

- ☒ **Sintaxe:** DbGoBotton()
- ☒ **Parâmetros**

Nenhum

.

Exemplo:

```
nCount := 0 // Variável para verificar quantos registros há no intervalo
DbSelectArea("SA1")
DbSetOrder(1) // A1_FILIAL + A1_COD + A1_LOJA
DbGoBotton()

While !EOF() // Enquanto não for o início do arquivo
    nCount++ // Incrementa a variável de controle de registros no intervalo
    DbSkip(1)
End

MsgInfo("Existem :"+STRZERO(nCount,6)+ "registros no intervalo").

// Retorno esperado :000001, pois o DbGoBotton posiciona no último registro.
```

DBINFO()

DBINFO() é utilizada para obter informações sobre a tabela corrente. O tipo de informação (primeiro argumento) é escolhido de acordo com as constantes abaixo:

- ☒ **Tabela A : Constantes utilizadas na parametrização da função**

Constante	Descrição	Retorno
DBI_GETRECSIZE	Tamanho do registro em número de bytes similar a RECSIZE.	Numérico
DBI_TABLEEXT	Extensão do arquivo da tabela corrente.	Caracter
DBI_FULLPATH	Nome da tabela corrente com caminho completo.	Caracter
DBI_BOF	Se está posicionada no início da tabela similar a BOF	Lógico
DBI_EOF	Se está posicionada no final da tabela similar a EOF	Lógico
DBI_FOUND	Se a tabela está posicionada após uma pesquisa similar a FOUND	Lógico
DBI_FCOUNT	Número de campos na estrutura da tabela corrente similar a FCOUNT	Numérico
DBI_ALIAS	Nome do Alias da área de trabalho corrente similar a ALIAS	Caracter
DBI_LASTUPDATE	Data da última modificação similar a LUPDATE	Data

☒ **Sintaxe:** DBINFO(<nINFOTIPO>)

☒ **Parâmetros:**

nINFOTIPO	Tipo de informação a ser verificada.
------------------	--------------------------------------

☒ **Retorno:**

Indefinido	Informação da Tabela Informação requisitada pelo usuário (o tipo depende da informação requisitada). Se não houver tabela corrente retorna NIL.
-------------------	---

Exemplo:

```
USE Clientes NEW
DBINFO(DBI_FULLPATH) // Retorno: C:\Teste\Clientes.dbf
DBINFO(DBI_FCOUNT) // Retorno: 12
DBGOTOP()
DBINFO(DBI_BOF) // Retorno: .F.
DBSKIP(-1)
DBINFO(DBI_BOF) // Retorno: .T.
```

DBNICKINDEXKEY()

Função que retorna o "IndexKey", ou seja, a expressão de índice da ordem especificada pelo NickName. Se não existe índice com o nickname, retorna uma string vazia.

☒ **Sintaxe:** DBNICKINDEXKEY(<cNick>)

☒ **Parâmetros:**

cNick	Indica o "NickName" da ordem de índice.
--------------	---

☒ **Retorno:**

Caracter	Expressão do índice identificado pelo "NickName".
-----------------	---



Anotações

DBORDERINFO()

A função DBORDERINFO() é utilizada para obter informações sobre determinada ordem. A especificação da ordem pode ser realizada através de seu nome ou sua posição dentro da lista de ordens, mas se ela não for especificada serão obtidas informações da ordem corrente. O tipo de informação (primeiro argumento) é escolhido de acordo com as constantes abaixo:

☒ **Tabela A : Constantes utilizadas na parametrização da função**

Constante	Descrição	Retorno
DBOI_BAGNAME	Nome do arquivo de índice ao qual a ordem pertence.	Caracter
DBOI_FULLPATH	do arquivo de índice (com seu diretório) ao qual a ordem pertence.	Caracter
DBOI_ORDERCOUNT	Número de ordens existentes no arquivo de índice especificado.	Caracter

☒ **Sintaxe: DBORDERINFO(<nINFOTIPO>)**

☒ **Parâmetros:**

nINFOTIPO	Nome do arquivo de índice.
------------------	----------------------------

☒ **Retorno:**

Caracter	Retorna a informação da Ordem requisitada pelo usuário (pode ser de tipo numérico se for número de ordens no índice, tipo caracter se for nome do arquivo de índice). Caso não exista ordem corrente ou a posição da ordem especificada está inválida retorna NIL.
-----------------	--

Exemplo:

DBORDERINFO(DBOI_BAGNAME) // retorna: Ind
DBORDERINFO(DBOI_FULLPATH) // retorna: C:\AP6\Teste\Ind.cdx



Anotações

DBORDERNICKNAME()

A função DBORDERNICKNAME() é utilizada para selecionar a ordem ativa através de seu apelido. Esta ordem é a responsável pela sequência lógica dos registros da tabela corrente.

☒ **Sintaxe:** DBORDERNICKNAME(<cAPELIDO>)

☒ **Parâmetros:**

cAPELIDO	Nome do apelido da ordem a ser setada.
-----------------	--

☒ **Retorno:**

Lógico	Retorna Falso se não conseguiu tornar a ordem ativa. Principais erros: Não existe tabela ativa ou não foi encontrada a ordem com o apelido. Retorna Verdadeiro se a ordem foi setada com sucesso.
---------------	---

Exemplo:

```
USE Cliente NEW
SET INDEX TO Nome, Idade

IF !DBORDERNICKNAME("IndNome")
    MessageBox("Registro não encontrado", "Erro", 0)
ENDIF
```

DBPACK()

A função DBPACK() remove fisicamente todos os registros com marca de excluído da tabela.

☒ **Sintaxe:** __DBPACK()

☒ **Parâmetros:**

Nenhum	.
---------------	---

☒ **Retorno:**

Nenhum	.
---------------	---

Exemplo:

```
USE Clientes NEW
DBGOTO(100)
DBDELETE()
DBGOTO(105)
DBDELETE()
DBGOTO(110)
DBDELETE()
```

```
// Se a exclusão for confirmada:
__DBPACK()
```


DBRECALL()

A função DBRECALL() é utilizada para retirar a marca de registro deletado do registro atual. Para ser executada o registro atual deve estar bloqueado ou a tabela deve estar aberta em modo exclusivo. Se o registro atual não estiver deletado, esta função não faz nada. Ela é o oposto da função DBDELETE() que marca o registro atual como deletado.

☒ **Sintaxe: DBRECALL()**

☒ **Parâmetros:**

Nenhum

.

☒ **Retorno:**

Nenhum

.

Exemplo 01: Desfazendo a deleção do registro posicionado do alias corrente

```
USE Cliente
DBGOTO(100)
DBDELETE()
DELETED() // Retorna: .T.
DBRECALL()
DELETED() // Retorna: .F.
```

Exemplo 02: Desfazendo as deleções do alias corrente

```
USE Cliente
DBGOTOP()
WHILE !EOF()
    DBRECALL()
    DBSKIP()
ENDDO
```

DBRECORDINFO()

A função DBRECORDINFO() é utilizada para obter informações sobre o registro especificado pelo segundo argumento (recno) da tabela corrente, se esta informação for omitida será verificado o registro corrente. O tipo de informação (primeiro argumento) é escolhido de acordo com as constantes abaixo:

☒ **Tabela A : Constantes utilizadas na parametrização da função**

Constante	Descrição	Retorno
DBRI_DELETED	Estado de deletado similar a DELETED	Lógico
DBRI_RECSIZE	Tamanho do registro similar a RECSIZE	Numérico
DBRI_UPDATED	Verifica se o registro foi alterado e ainda não foi atualizado fisicamente similar a UPDATED	Lógico

☑ **Sintaxe:** DBRECORDINFO (< nINFOTIPO > , [nREGISTRO]) --> xINFO

☑ **Parâmetros:**

nINFOTIPO	Tipo de informação a ser verificada.
nREGISTRO	Número do registro a ser verificado.

☑ **Retorno:**

Indefinido	Não há tabela corrente ou registro inválido. Informação do Registro. Informação requisitada pelo usuário (o tipo depende da informação requisitada).
-------------------	--

Exemplo:

```
USE Clientes NEW
DBGOTO(100)
DBRECORDINFO(DBRI_DELETED) // Retorno: .F.
DBDELETE()
DBRECORDINFO(DBRI_DELETED) // Retorno: .F.
DBRECALL()
DBRECORDINFO(DBRI_RECSIZE) // Retorno: 230
NOME := "JOAO"
DBGOTO(200)
DBRECORDINFO(DBRI_UPDATED) // Retorno: .F.
DBRECORDINFO(DBRI_UPDATED,100) // Retorno: .T.
```

DBREINDEX()

A função DBREINDEX() reconstrói todos os índices da área de trabalho corrente e posiciona as tabelas no primeiro registro lógico.

☑ **Sintaxe:** DBREINDEX()

☑ **Parâmetros:**

Nenhum	.
---------------	---

☑ **Retorno:**

Nenhum	.
---------------	---

Exemplo:

```
USE Clientes NEW
DBSETINDEX("IndNome")
DBREINDEX()
```

DBRLOCK()

A função DBRLOCK() é utilizada quando se tem uma tabela aberta e compartilhada e se deseja bloquear um registro para que outros usuários não possam alterá-lo. Se a tabela já está aberta em modo exclusivo, a função não altera seu estado. O usuário pode escolher o registro a ser bloqueado através do parâmetro (recno), mas se este for omitido será bloqueado o registro corrente como na função RLOCK(). Esta função é o oposto à DBRUNLOCK, que libera registros bloqueados.

☑ **Sintaxe:** DBRLOCK([nREGISTRO])

☑ **Parâmetros:**

nREGISTRO	Número do registro a ser bloqueado.
-----------	-------------------------------------

☑ **Retorno:**

Lógico	Retorna Falso se não conseguiu bloquear o registro. Principal motivo: o registro já foi bloqueado por outro usuário.
	Retorna Verdadeiro se o registro foi bloqueado com sucesso.

Exemplo:

```
DBUSEAREA( .T., "dbfcxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F. )
DBGOTO(100)
DBRLOCK() // Bloqueia o registro atual (100)
DBRLOCK(110) // Bloqueia o registro de número 110
```

DBRLOCKLIST()

A função DBRLOCKLIST() é utilizada para verificar quais registros estão locados na tabela corrente. Para tanto, é retornada uma tabela unidimensional com os números dos registros.

☑ **Sintaxe:** DBRLOCKLIST()

☑ **Parâmetros:**

Nenhum	.
--------	---

☑ **Retorno:**

Array	Retorna NIL se não existe tabela corrente ou não existe nenhum registro locado. Retorna a lista com os recnos dos registros locados na tabela corrente.
--------------	---

Exemplo:

```
DBUSEAREA( .T., "dbfcdxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F. )
DBGOTOP()
DBRLOCK() // Bloqueia o primeiro registro
DBRLOCK(110) // Bloqueia o registro de número 110
DBRLOCK(100) // Bloqueia o registro de número 100
DBRLOCKLIST() // Retorna: {1,100,110}
```

DBRUNLOCK()

A função DBRUNLOCK() é utilizada para liberar determinado registro bloqueado. O usuário pode escolher o registro a ser desbloqueado através do parâmetro (Recno), mas se este for omitido será desbloqueado o registro corrente como na função DBUNLOCK(). Esta função é o oposto à DBRLOCK, que bloqueia os registros.

☒ **Sintaxe:** DBRUNLOCK([nREGISTRO])

☒ **Parâmetros:**

nREGISTRO	Número do registro a ser desbloqueado.
------------------	--

☒ **Retorno:**

Nenhum	.
---------------	---

Exemplo:

```
DBUSEAREA( .T., "dbfcdxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F. )
DBGOTO(100)
DBRUNLOCK() // Desbloqueia o registro atual (100)
DBRUNLOCK(110) // Desbloqueia o registro de número 110
```

DBSETDRIVER()

A função DBSETDRIVER() pode ser utilizada apenas para verificar qual o RDD que está definido como padrão quando for omitido seu parâmetro. Ela também pode ser utilizada para especificar outro RDD como padrão, especificando-o através do parâmetro.

☒ **Sintaxe:** DBSETDRIVER([cNOVORDD])

☒ **Parâmetros:**

cNOVORDD	Novo nome do RDD a ser definido como padrão.
-----------------	--

☒ **Retorno:**

Caracter	Nome do RDD padrão corrente.
-----------------	------------------------------

Exemplo:

```
DBSETDRIVER("CTREECDX") // Retorna: DBFCDX  
DBSETDRIVER() // Retorna: CTREECDX
```

**Importante**

Note que ao utilizar a função DBSETDRIVER para redefinir o driver corrente, o retorno da função não será o driver definido nos parâmetros, mas o driver que estava em uso antes da atualização.

DBSETINDEX()

A função DBSETINDEX() é utilizada para acrescentar uma ou mais ordens de determinado índice na lista de ordens ativas da área de trabalho. Quando o arquivo de índice possui apenas uma ordem, a mesma é acrescentada à lista e torna-se ativa. Quando o índice possui mais de uma ordem, todas são acrescentadas à lista e a primeira torna-se ativa.

**Importante**

Para utilizar os arquivos de extensão padrão do RDD este dado deve ser especificado.

- ☒ **Sintaxe:** DBSETINDEX(<@cARQINDICE>)
- ☒ **Parâmetros:**

cARQINDICE	Nome do arquivo de índice, com ou sem diretório.
-------------------	--

- ☒ **Retorno:**

Nenhum	.
---------------	---

Exemplo:

```
USE Cliente NEW  
DBSETINDEX("Ind1")  
DBSETINDEX("\teste\Ind2.cdx")
```

**Anotações**

DBSETNICKNAME()

A função DBSETNICKNAME() é utilizada para colocar um apelido em determinada ordem especificada pelo primeiro parâmetro. Caso seja omitido o nome do apelido a ser dado, a função apenas verifica o apelido corrente.

☑ **Sintaxe:** DBSETNICKNAME(<cINDICE>, [cAPELIDO])

☑ **Parâmetros:**

cINDICE	Nome da ordem que deve receber o apelido.
cAPELIDO	Nome do apelido da ordem a ser setada.

☑ **Retorno:**

Caracter	Retorna "" (String vazia) se não conseguiu encontrar a ordem especificada, não conseguiu setar o apelido ou não havia apelido. Retorna o apelido corrente.
----------	--

Exemplo:

```
USE Cliente NEW
DBSETNICKNAME("IndNome") // retorna: ""
DBSETNICKNAME("IndNome","NOME") // retorna: ""
DBSETNICKNAME("IndNome") // retorna: "NOME"
```

DBSELECTAREA()

Define a área de trabalho especificada com sendo a área ativa. Todas as operações subseqüentes que fizerem referência a uma área de trabalho a utilização, a menos que a área desejada seja informada explicitamente.

☑ **Sintaxe:** DbSelectArea(nArea | cArea)

☑ **Parâmetros**

nArea	Valor numérico que representa a área desejada, em função de todas as áreas já abertas pela aplicação, que pode ser utilizado ao invés do nome da área.
cArea	Nome de referência da área de trabalho a ser selecionada.

Exemplo 01: DbselectArea(nArea)

```
nArea := Select("SA1") // → 10 (proposto)
```

```
DbSelectArea(nArea) // De acordo com o retorno do comando Select()
```

```
ALERT("Nome do cliente: "+A1_NOME) // Como o SA1 é o alias selecionado, os comandos
// a partir da seleção do alias compreendem que ele
// está implícito na expressão, o que causa o mesmo
// efeito de SA1->A1_NOME
```

Exemplo 02: DbselectArea(cArea)

DbSelectArea("SA1") // Especificação direta do alias que deseja-se selecionar

ALERT("Nome do cliente: "+A1_NOME) // Como o SA1 é o alias selecionado, os comandos
// a partir da seleção do alias compreendem que ele
// está implícito na expressão, o que causa o mesmo
// efeito de SA1->A1_NOME

DBSETORDER()

Define qual índice será 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 SINDEIX /SIX, ou as ordens disponibilizadas por meio de índices temporários.

☑ **Sintaxe: DbSetOrder(nOrdem)**

☑ **Parâmetros**

nOrdem	Número de referência da ordem que deseja ser definida como ordem ativa para a área de trabalho.
---------------	---

Exemplo:

DbSelectArea("SA1")
DbSetOrder(1) // De acordo com o arquivo SIX -> A1_FILIAL+A1_COD+A1_LOJA

DBORDERNICKNAME()

Define qual índice criado pelo usuário seja utilizado. O usuário pode incluir os seus próprios índices e no momento da inclusão deve criar o NICKNAME para o mesmo.

☑ **Sintaxe: DbOrderNickName(NickName)**

☑ **Parâmetros**

NickName	NickName atribuído ao índice criado pelo usuário
-----------------	--

Exemplo:

DbSelectArea("SA1")
DbOrderNickName("Tipo") // De acordo com o arquivo SIX -> A1_FILIAL+A1_TIPO
NickName: Tipo

DBSEEK() E MSSEEK()

DbSeek(): Permite posicionar o cursor da área de trabalho ativo no registro com as informações especificadas na chave de busca, fornecendo um retorno lógico indicando se o posicionamento foi efetuado com sucesso, ou seja, se a informação especificada na chave de busca foi localizada na área de trabalho.

- ☑ **Sintaxe:** DbSeek(cChave, ISoftSeek, ILast)
- ☑ **Parâmetros**

cChave	Dados do registro que deseja-se localizar, de acordo com a ordem de busca previamente especificada pelo comando DbSetOrder(), ou seja, de acordo com o índice ativo no momento para a área de trabalho.
ISoftSeek	Define se o cursor ficará posicionado no próximo registro válido, em relação a chave de busca especificada, ou em final de arquivo, caso não seja encontrada exatamente a informação da chave. Padrão → .F.
ILast	Define se o cursor será posicionado no primeiro ou no último registro de um intervalo com as mesmas informações especificadas na chave. Padrão → .F.

Exemplo 01 – Busca exata

```
DbSelectArea("SA1")
DbSetOrder(1) // acordo com o arquivo SIX -> A1_FILIAL+A1_COD+A1_LOJA

IF DbSeek("01" + "000001" + "02" ) // Filial: 01, Código: 000001, Loja: 02
    MsgInfo("Cliente localizado", "Consulta por cliente")
Else
    MsgAlert("Cliente não encontrado", "Consulta por cliente")
Endif
```

Exemplo 02 – Busca aproximada

```
DbSelectArea("SA1")
DbSetOrder(1) // acordo com o arquivo SIX -> A1_FILIAL+A1_COD+A1_LOJA

DbSeek("01" + "000001" + "02", .T. ) // Filial: 01, Código: 000001, Loja: 02

// Exibe os dados do cliente localizado, o qual pode não ser o especificado na chave:
MsgInfo("Dados do cliente localizado: "+CRLF +;
    "Filial:" + A1_FILIAL + CRLF +;
    "Código:" + A1_COD + CRLF +;
    "Loja:" + A1_LOJA + CRLF +;
    "Nome:" + A1_NOME + CRLF, "Consulta por cliente")
```


MsSeek(): Função desenvolvida pela área de Tecnologia da Microsiga, a qual possui as mesmas funcionalidades básicas da função DbSeek(), com a vantagem de não necessitar acessar novamente a base de dados para localizar uma informação já utilizada pela thread (conexão) ativa.

Desta forma, a thread mantém em memória os dados necessários para reposicionar os registros já localizados através do comando DbSeek (no caso o Recno()) de forma que a aplicação pode simplesmente efetuar o posicionamento sem executar novamente a busca.

A diferença entre o DbSeek() e o MsSeek() é notada em aplicações com grande volume de posicionamentos, como relatórios, que necessitam referenciar diversas vezes o mesmo registro durante uma execução.

DBSKIP()

Move o cursor do registro posicionado para o próximo (ou anterior dependendo do parâmetro), em função da ordem ativa para a área de trabalho.

- ☒ **Sintaxe: DbSkip(nRegistros)**
- ☒ **Parâmetros**

nRegistros	Define em quantos registros o cursor será deslocado. Padrão → 1
-------------------	---

Exemplo 01 – Avançando registros

```
DbSelectArea("SA1")
DbSetOrder(2) // A1_FILIAL + A1_NOME
DbGotop() // Posiciona o cursor no início da área de trabalho ativa

While !EOF() // Enquanto o cursor da área de trabalho ativa não indicar fim de arquivo
    MsgInfo("Você está no cliente:" + A1_NOME)
    DbSkip()
End
```

Exemplo 02 – Retrocedendo registros

```
DbSelectArea("SA1")
DbSetOrder(2) // A1_FILIAL + A1_NOME
DbGoBottom() // Posiciona o cursor no final da área de trabalho ativa

While !BOF() // Enquanto o cursor da área de trabalho ativa não indicar início de arquivo
    MsgInfo("Você está no cliente:" + A1_NOME)
    DbSkip(-1)
End
```

DBSETFILTER()

Define um filtro para a área de trabalho ativa, o qual pode ser descrito na forma de um bloco de código ou através de uma expressão simples.

- ☑ **Sintaxe:** `DbSetFilter(bCondicao, cCondicao)`
- ☑ **Parâmetros**

bCondicao	Bloco de expressa a condição de filtro em forma executável
cCondicao	Expressão de filtro simples na forma de string

Exemplo 01 – Filtro com bloco de código

```
bCondicao := {|| A1_COD >= "000001" .AND. A1_COD <= "001000"}
DbSelectArea("SA1")
DbSetOrder(1)
DbSetFilter(bCondicao)
DbGoBotton()

While !EOF()
    MsgInfo("Você está no cliente:" + A1_COD)
    DbSkip()
End

// O último cliente visualizado deve ter o código menor do que "001000".
```

Exemplo 02 – Filtro com expressão simples

```
cCondicao := "A1_COD >= '000001' .AND. A1_COD <= '001000'"
DbSelectArea("SA1")
DbSetOrder(1)
DbSetFilter(cCondicao)
DbGoBotton()

While !EOF()
    MsgInfo("Você está no cliente:" + A1_COD)
    DbSkip()
End

// O último cliente visualizado deve ter o código menor do que "001000".
```

DBSTRUCT()

Retorna um array contendo a estrutura da área de trabalho (alias) ativo. A estrutura será um array bidimensional conforme abaixo:

ID*	Nome campo	Tipo campo	Tamanho	Decimais
-----	------------	------------	---------	----------

*Índice do array

- ☒ **Sintaxe:** DbStruct()
- ☒ **Parâmetros**

Nenhum	.
--------	---

Exemplo:

```
cCampos := ""
DbSelectArea("SA1")
aStructSA1 := DbStruct()

FOR nX := 1 to Len(aStructSA1)
    cCampos += aStructSA1[nX][1] + "/"
NEXT nX

ALERT(cCampos)
```

DBUNLOCK()

A função DBUNCLOCK() retira os bloqueios dos registros e do arquivo da tabela corrente.

- ☒ **Sintaxe:** DBUNLOCK()
- ☒ **Parâmetros:**

Nenhum	.
--------	---

- ☒ **Retorno:**

Nenhum	.
--------	---

DBUNLOCKALL()

A função DBUNLOCKALL() Retira os bloqueios de todos os registros e dos arquivos de todas as tabelas abertas. Esta função é utilizada para liberar todos os registros bloqueados e é equivalente a executar DBUNLOCK para todas as tabelas da área de trabalho.

☒ **Sintaxe:** DBUNLOCKALL()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Nenhum	.
--------	---

DBUSEAREA()

Define um arquivo de base de dados como uma área de trabalho disponível na aplicação.

☒ **Sintaxe:** DbUseArea(INovo, cDriver, cArquivo, cAlias, IComparilhado,; ISoLeitura)

☒ **Parâmetros**

INovo	Parâmetro opcional que permite que se caso o cAlias especificado já esteja em uso, ele seja fechado antes da abertura do arquivo da base de dados.
cDriver	Driver que permita a aplicação manipular o arquivo de base de dados especificado. A aplicação ERP possui a variável __LOCALDRIVER definida a partir das configurações do .ini do server da aplicação. Algumas chaves válidas: "DBFCDX", "CTREECDX", "DBFCDXAX", "TOPCONN".
cArquivo	Nome do arquivo de base de dados que será aberto com o alias especificado.
cAlias	Alias para referência do arquivos de base de dados pela aplicação.
IComparilhado	Se o arquivo poderá ser utilizado por outras conexões.
ISoLeitura	Se o arquivo poderá ser alterado pela conexão ativa.

Exemplo:

```
DbUserArea(.T., "DBFCDX", "\\SA1010.DBF", "SA1DBF", .T., .F.)
DbSelectArea("SA1DBF")
MsgInfo("A tabela SA1010.DBF possui:" + STRZERO(RecCount(),6) + " registros.")
DbCloseArea()
```

DELETED()

A função DELETED() Verifica se o registro está com marca de excluído. Quando o registro é excluído, permanece fisicamente na tabela, mas fica marcado como excluído. Esta função verifica este estado. Se nenhuma área está selecionada, retorna .F.. Quando é executada a função DBPACK() todos os registros marcados como deletados são apagados fisicamente. A função DBRECALL() retira todas as marcas.

☒ **Sintaxe:** DELETED()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Nenhum	.
--------	---

Exemplo:

```
USE "\DADOSADV\AA1990.DBF" SHARED NEW
DBGOTO(100)
IF DELETED()
  MessageBox("O registro atual foi deletado","Erro", 0)
ENDIF
```

FCOUNT()

A função FCOUNT() avalia a quantidade de campos existentes na estrutura do arquivo ativo como área de trabalho.

☒ **Sintaxe:** FCOUNT()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Númérico	Quantidade de campos existentes na estrutura da área de trabalho ativa.
----------	---

Exemplo:

```
DbSelectArea("SA1")
nFields := FCOUNT()

IF nFields > 0
  MSGINFO("A estrutura da tabela contém :+CvalToChar(nFields)+"campos.")
ENDIF
```

FOUND()

A função FOUND() recupera o resultado de sucesso referente a última operação de busca efetuada pelo processamento corrente.

☑ **Sintaxe: FOUND()**

☑ **Parâmetros:**

Nenhum	.
--------	---

☑ **Retorno:**

Lógico	Indica se a última operação de busca realizada pelo processamento corrente obteve sucesso (.T.) ou não (.F.).
--------	---

Exemplo:

```
Pergunte(cPerg,.T.)
DbSelectArea("SA1")
DbSetOrder(1)
DbSeek(xFilial("SA1")+MVPAR01)

IF Found()
    MSGINFO("Cliente encontrado")
ELSE
    MSGALERT("Dados não encontrados")
ENDIF
```

INDEXKEY()

A função INDEXKEY() determina a expressão da chave de um índice especificado na área de trabalho corrente, e o retorna na forma de uma cadeia de caracteres, sendo normalmente utilizada na área de trabalho correntemente selecionada.

☑ **Sintaxe: INDEXKEY()**

☑ **Parâmetros:**

nOrdem	Ordem do índice na lista de índices abertos pelo comando USE...INDEX ou SET INDEX TO na área de trabalho corrente. O valor default zero especifica o índice corrente, independentemente de sua posição real na lista.
--------	---

☑ **Retorno:**

Caracter	Expressão da chave do índice especificado na forma de uma cadeia de caracteres. Caso não haja um índice correspondente, INDEXKEY() retorna uma cadeia de caracteres vazia ("").
----------	---

Exemplo:

```
cExpressao := SA1->(IndexKey())
```

INDEXORD()

A função INDEXORD() verifica a posição do índice corrente na lista de índices do respectivo alias.

☑ **Sintaxe:** INDEXORD()

☑ **Parâmetros:**

Nenhum	.
--------	---

☑ **Retorno:**

Numérico	Posição do índice corrente na lista de índices da tabela. Retorna 0 se não existe índice aberto na tabela corrente.
----------	---

Exemplo:

```
USE Cliente NEW
SET INDEX TO Nome, End, Cep
nOrd:=INDEXORD() // Return: 1 - é o primeiro índice da lista
```

LUPDATE()

A função LUPDATE() verifica qual a data da última modificação e fechamento da tabela corrente, sendo que caso não exista tabela corrente é retornada uma data em branco.

☑ **Sintaxe:** LUPDATE()

☑ **Parâmetros:**

Nenhum	.
--------	---

☑ **Retorno:**

Data	Retorna um valor do tipo Data , indicando a data da ultima modificação e fechamento da Tabela. Caso não haja tabela selecionada na área de trabalho atual , a função retornará uma data vazia (ctod("")) .
------	--

Exemplo:

```
// Mostra a data da última modificação da tabela corrente,
dModificacao := LUpdate()
IF (EMPTY(dModificacao))
    CONOUT("Não há tabela corrente")
ELSE
    CONOUT(("Data da ultima modificacao : " + DTOS(dModificacao)))
ENDIF
```

MSAPPEND()

A função MsAppend() adiciona registros de um arquivo para outro, respeitando a estrutura das tabelas.

☑ **Sintaxe:** MSAPPEND([cArqDest], cArqOrig)

☑ **Parâmetros:**

cArqDest	Se o RDD corrente for DBFCDX os registros serão adicionados na área selecionada, caso contrário o arquivo destino terá que ser informado.
cArqOrig	Nome do arquivo origem contendo os registros a serem adicionados.

☑ **Retorno:**

Lógico	Se a operação for realizada com sucesso o função retornará verdadeiro (.T.).
---------------	--

Exemplo:

```
dbSelectArea('XXX')  
MsAppend('ARQ00001')
```

MSUNLOCK()

Libera o travamento (lock) do registro posicionado confirmando as atualizações efetuadas neste registro.

☑ **Sintaxe:** MsUnLock()

☑ **Parâmetros**

Nenhum	.
---------------	---

Exemplo:

```
DbSelectArea("SA1")  
DbSetOrder(1) // A1_FILIAL + A1_COD + A1_LOJA  
DbSeek("01" + "900001" + "01") // Busca exata  
  
IF Found() // Avalia o retorno do último DbSeek realizado  
RecLock("SA1",.F.)  
SA1->A1_NOME := "CLIENTE CURSO ADVPL BÁSICO"  
SA1->A1_NREDUZ := "ADVPL BÁSICO"  
MsUnLock() // Confirma e finaliza a operação  
ENDIF
```


ORDBAGEXT()

A função ORDBAGEXT é utilizada no gerenciamento de índices para os arquivos de dados do sistema, permitindo avaliar qual a extensão deste índices atualmente em uso, de acordo com a RDD ativa.

☑ **Sintaxe:** ORDBAGEXT()

☑ **Parâmetros:**

Nenhum	.
--------	---

☑ **Retorno:**

cBagExt	Extensão do arquivo dos arquivos de índices em uso pelo sistema, determinado pela RDD ativa.
---------	--

Exemplo:

```
cArqTRB := CriaTrab(aStruTRB,.T.)
// RDD UTILIZADA: "DBFCDXADS"
DbUseArea(.T., "DBFCDXADS", cArqTRB, "TRBSA1", .F., .F.)

DbSelectArea("TRBSA1")
cArqInd := CriaTrab(Nil,.F.)
IndRegua("TRBSA1",cArqInd,cChaveInd,,"","Selecionando registros ...")
#IFDEF TOP
    DbSetIndex(cArqInd+OrdBagExt())
    // RETORNO: ".CDX"
#ENDIF
DbSetOrder(1)
```

ORDKEY()

A função ORDKEY() verifica qual é a expressão de chave de determinada ordem. Caso não sejam especificados os parâmetros de identificação da ordem, é verificada a ordem corrente. Para evitar conflito, no caso de haver mais de uma ordem com o mesmo nome, pode-se passar o parâmetro com o nome do índice ao qual a ordem pertence.

A ordem passada no primeiro parâmetro pode ser especificada através da sua posição na lista de ordens ativas (através do ORDLISTADD) ou através do nome dado à ordem. A função verifica automaticamente se o parâmetro é numérico ou caracter.

☑ **Sintaxe:** ORDKEY([cOrdem | nPosicao] , [cArqIndice])

☑ **Parâmetros:**

cOrdem	Há duas opções para o primeiro parâmetro:
nPosicao	cNome: tipo caracter, contém nome do índice. nPosicao: tipo numérico, indica ordem do índice.
cArqIndice	Nome do arquivo de índice.

☒ **Retorno:**

Caracter	Expressão de chave da ordem ativa ou especificada pelos parâmetros. Cadeia vazia indica que não existe ordem corrente.
-----------------	--

Exemplo:

```
USE Cliente NEW
INDEX ON Nome+Cod TO Ind1 FOR Nome+Cod > 'AZZZZZZZ'
ORDKEY('Ind1')
// Retorna: Nome+Cod
```

RECLOCK()

Efetua o travamento do registro posicionado na área de trabalho ativa, permitindo a inclusão ou alteração das informações do mesmo.

☒ **Sintaxe: RecLock(cAlias,II Inclui)**

☒ **Parâmetros**

cAlias	Alias que identifica a área de trabalho que será manipulada.
II Inclui	Define se a operação será uma inclusão (.T.) ou uma alteração (.F.)

Exemplo 01 - Inclusão

```
DbSelectArea("SA1")
RecLock("SA1",.T.)
SA1->A1_FILIAL := xFilial("SA1") // Retorna a filial de acordo com as configurações do ERP
SA1->A1_COD := "900001"
SA1->A1_LOJA := "01"
MsUnLock() // Confirma e finaliza a operação
```

Exemplo 02 - Alteração

```
DbSelectArea("SA1")
DbSetOrder(1) // A1_FILIAL + A1_COD + A1_LOJA
DbSeek("01" + "900001" + "01") // Busca exata

IF Found() // Avalia o retorno do último DbSeek realizado
RecLock("SA1",.F.)
SA1->A1_NOME := "CLIENTE CURSO ADVPL BÁSICO"
SA1->A1_NREDUZ := "ADVPL BÁSICO"
MsUnLock() // Confirma e finaliza a operação
ENDIF
```



Dica

A linguagem ADVPL possui variações da função RecLock(), as quais são:

- ☒ RLOCK()
- ☒ DBRLOCK()

A sintaxe e a descrição destas funções estão disponíveis no Guia de Referência Rápido ao final deste material.



Dica

A linguagem ADVPL possui variações da função MsUnlock(), as quais são:

- ☒ UNLOCK()
- ☒ DBUNLOCK()
- ☒ DBUNLOCKALL()

A sintaxe e a descrição destas funções estão disponíveis no Guia de Referência Rápido ao final deste material.

RECNO()

A função RECNO() retorna o número do registro atualmente posiconado na área de trabalho ativa.

☒ **Sintaxe:** RECNO()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

nRecno	Identificador numérico do registro atualmente posicionando na área de trabalho ativa.
--------	---

Exemplo:

```
DbSelectArea("SA1")
DbGoto(100) // Posiciona no registro de recno 100.
MSGINFO("Registro posicionado:"+cValToChar(RECNO()))
```

SELECT()

A função SELECT() determina o número da área de trabalho de um alias. O número retornado pode variar de zero a 250. Se <cAlias> não for especificado, é retornado o número da área de trabalho corrente. Caso <cAlias> seja especificado e o alias nao existir, SELECT() retorna zero.

☒ **Sintaxe:** SELECT([cAlias])

☒ **Parâmetros:**

cAlias	Nome da área de trabalho a ser verificada.
--------	--

☒ **Retorno:**

Numérico	Área de trabalho do alias especificado na forma de um valor numérico inteiro.
----------	---

Exemplo:

```
nArea := Select("SA1")
```

```
ALERT("Referência do alias SA1: "+STRZERO(nArea,3)) // → 10 (proposto)
```

SET FILTER TO

O comando SET FILTER TO define uma condição de filtro que será aplicada a área de trabalho ativa.



Recomenda-se o uso da função DbSetFilter() em substituição ao comando SET FILTER TO

☒ **Sintaxe: SET FILTER TO cCondicao**

☒ **Parâmetros:**

cCondicao	Expressão que será avaliada pela SET FILTER, definindo os registros que ficarão disponíveis na área de trabalho ativa. Esta expressão obrigatoriamente deve ter um retorno lógico.
------------------	---

☒ **Retorno:**

Nenhum	.
---------------	---



O uso da sintaxe SET FILTER TO desativa o filtro na área de trabalho corrente.

Exemplo:

```
USE Employee INDEX Name NEW
```

```
SET FILTER TO Age > 50
```

```
LIST LastName, FirstName, Age, Phone
```

```
SET FILTER TO
```

SOFTLOCK()

Permite a reserva do registro posicionado na área de trabalho ativa de forma que outras operações, com exceção da atual, não possam atualizar este registro. Difere da função RecLock() pois não gera uma obrigação de atualização, e pode ser sucedido por ele.

Na aplicação ERP Protheus, o SoftLock() é utilizado nos browses, antes da confirmação da operação de alteração e exclusão, pois neste momento a mesma ainda não foi efetivada, mas outras conexões não podem acessar aquele registro pois o mesmo está em manutenção, o que implementa a integridade da informação.

☒ **Sintaxe: SoftLock(cAlias)**

☒ **Parâmetros**

cAlias	Alias de referência da área de trabalho ativa, para o qual o registro posicionado será travado.
---------------	---

Exemplo:

```
cChave := GetCliente() // Função ilustrativa que retorna os dados de busca de um cliente

DbSelectArea("SA1")
DbSetOrder(1)
DbSeek(cChave)

IF Found()
    SoftLock() // Reserva o registro localizado
    IConfirma := AlteraSA1() // Função ilustrativa que exibe os dados do registro
                                // posicionado e permite a alteração dos mesmos.

    IF IConfirma
        RecLock("SA1",.F.)
        GravaSA1() // Função ilustrativa que altera os dados conforme a AlteraSA1()
        MsUnLock() // Liberado o RecLock() e o SoftLock() do registro.
    Endif
Endif
```



Anotações

USED()

A função USED() é utilizada para determinar se há um arquivo de banco de dados em uso em uma área de trabalho específica. O padrão é que USED() opere na área de trabalho correntemente selecionada.

☒ **Sintaxe: USED()**

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Lógico	Verdadeiro (.T.) caso haja um arquivo de banco de dados em uso; caso contrário, retorna falso (.F.).
--------	--

Exemplo:

```
USE Customer NEW
CONOUT(USED()) // Resulta: .T.
CLOSE
CONOUT (USED()) // Resulta: .F.
```

ZAP

O comando ZAP remove fisicamente todos os registro da tabela corrente. É necessário que o alias em questão seja aberto em modo exclusivo para esta operação ser realizada.

☒ **Sintaxe: ZAP**

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Nenhum	.
--------	---

Exemplo:

```
cTabela := RetSqlName("SA4")
cAlias := "TMP"
USE (cTabela) ALIAS (cAlias) EXCLUSIVE NEW VIA "TOPCONN"
IfNetErr()
  MsgStop("Nao foi possivel abrir "+cTabela+" em modo EXCLUSIVO.")
Else
  ZAP
  USE
  MsgStop("Registros da tabela "+cTabela+" eliminados com sucesso.")
Endif
```

Controle de numeração seqüencial

GETSXENUM()

Obtém o número seqüência do alias especificado no parâmetro, através da referência aos arquivos de sistema SXE/SXF ou ao servidor de numeração, quando esta configuração está habilitada no ambiente Protheus.

- ☑ **Sintaxe: GETSXENUM(cAlias, cCampo, cAliasSXE, nOrdem)**
- ☑ **Parâmetros**

cAlias	Alias de referência da tabela para a qual será efetuado o controle da numeração seqüencial.
cCampo	Nome do campo no qual está implementado o controle da numeração.
cAliasSXE	Parâmetro opcional, quando o nome do alias nos arquivos de controle de numeração não é o nome convencional do alias para o sistema ERP.
nOrdem	Número do índice para verificar qual a próxima ocorrência do número.

CONFIRMSXE()

Confirma o número alocado através do último comando GETSXENUM().

- ☑ **Sintaxe: CONFIRMSXE(IVerifica)**
- ☑ **Parâmetros**

IVerifica	Verifica se o número confirmado não foi alterado, e por conseqüência já existe na base de dados.
------------------	--

ROLLBACKSXE()

Descarta o número fornecido pelo último comando GETSXENUM(), retornando a numeração disponível para outras conexões.

- ☑ **Sintaxe: ROLLBACKSXE()**
- ☑ **Parâmetros**

Nenhum	.
---------------	---



Anotações

Validação

ALLWAYSFALSE()

A função AllwaysFalse() foi criada com o objetivo de compatibilidade, sendo que sempre irá retornar um valor lógico falso, facilitando a especificação desta situação nas parametrizações de validações de modelos de interface pré-definidos no sistema.

☒ **Sintaxe:** ALLWAYSFALSE()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Lógico	Retorna um valor lógico sempre falso.
--------	---------------------------------------

ALLWAYSTRUE()

A função AllwaysTrue() foi criada com o objetivo de compatibilidade, sendo que sempre irá retornar um valor lógico verdadeiro, facilitando a especificação desta situação nas parametrizações de validações de modelos de interface pré-definidos no sistema.

☒ **Sintaxe:** ALLWAYSTRUE()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Lógico	Retorna um valor lógico sempre verdadeiro.
--------	--

EXISTCHAV()

Retorna .T. ou .F. se o conteúdo especificado existe no alias especificado. Caso exista será exibido um help de sistema com um aviso informando da ocorrência.

Função utilizada normalmente para verificar se um determinado código de cadastro já existe na tabela na qual a informação será inserida, como por exemplo o CNPJ no cadastro de clientes ou fornecedores.

☒ **Sintaxe:** ExistChav(cAlias, cConteudo, nIndice)

☒ **Parâmetros**

cAlias	Alias de referência para a validação da informação.
cConteudo	Chave a ser pesquisada, sem a filial.
nIndice	Índice de busca para consulta da chave.

EXISTCPO()

Retorna .T. ou .F. se o conteúdo especificado não existe no alias especificado. Caso não exista será exibido um help de sistema com um aviso informando da ocorrência.

Função utilizada normalmente para verificar se a informação digitada em um campo, a qual depende de outra tabela, realmente existe nesta outra tabela, como por exemplo o código de um cliente em um pedido de venda.

- ☒ **Sintaxe: ExistCpo(cAlias, cConteudo, nIndice)**
- ☒ **Parâmetros**

cAlias	Alias de referência para a validação da informação.
cConteudo	Chave a ser pesquisada, sem a filial.
nIndice	Índice de busca para consulta da chave.

LETTERORNUM()

A função LETTERORNUM() avalia se um determinado conteúdo é composto apenas de letras e números (alfanumérico).

- ☒ **Sintaxe: LETTERORNUM(cString)**
- ☒ **Parâmetros:**

cString	String que terá seu conteúdo avaliado.
----------------	--

- ☒ **Retorno:**

Lógico	Indica que se a string avaliada contém apenas letras e número, ou seja, alfanumérico.
---------------	---

NAOVAZIO()

Retorna .T. ou .F. se o conteúdo do campo posicionado no momento não está vazio.

- ☒ **Sintaxe: NaoVazio()**
- ☒ **Parâmetros**

Nenhum	.
---------------	---

NEGATIVO()

Retorna .T. ou .F. se o conteúdo digitado para o campo é negativo.

- ☒ **Sintaxe: Negativo()**
- ☒ **Parâmetros**

Nenhum	.
---------------	---

PERTENCE()

Retorna .T. ou .F. se o conteúdo digitado para o campo está contido na string definida como parâmetro da função. Normalmente utilizada em campos com a opção de combo, pois caso contrário seria utilizada a função ExistCpo().

- ☒ **Sintaxe: Pertence(cString)**
- ☒ **Parâmetros**

cString	String contendo as informações válidas que podem ser digitadas para um campo.
----------------	---

POSITIVO()

Retorna .T. ou .F. se o conteúdo digitado para o campo é positivo.

- ☒ **Sintaxe: Positivo()**
- ☒ **Parâmetros**

Nenhum	.
---------------	---

TEXTO()

Retorna .T. ou .F. se o conteúdo digitado para o campo contém apenas números ou alfanuméricos.

- ☒ **Sintaxe: Texto()**
- ☒ **Parâmetros**

Nenhum	.
---------------	---

VAZIO()

Retorna .T. ou .F. se o conteúdo do campo posicionado no momento está vazio.

- ☒ **Sintaxe: Vazio()**
- ☒ **Parâmetros**

Nenhum	.
---------------	---



Anotações

Manipulação de parâmetros do sistema

GETMV()

Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Caso o parâmetro não exista será exibido um help do sistema informando a ocorrência.

- ☒ **Sintaxe: GETMV(cParametro)**
- ☒ **Parâmetros**

cParametro	Nome do parâmetro do sistema no SX6, sem a especificação da filial de sistema.
-------------------	--

GETNEWPAR()

Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Caso o parâmetro não exista será exibido um help do sistema informando a ocorrência.

Difere do SuperGetMV() pois considera que o parâmetro pode não existir na versão atual do sistema, e por consequência não será exibida a mensagem de help.

- ☒ **Sintaxe: GETNEWPAR(cParametro, cPadrao, cFilial)**
- ☒ **Parâmetros**

cParametro	Nome do parâmetro do sistema no SX6, sem a especificação da filial de sistema.
cPadrao	Conteúdo padrão que será utilizado caso o parâmetro não exista no SX6.
cFilial	Define para qual filial será efetuada a consulta do parâmetro. Padrão → filial corrente da conexão.



Anotações

PUTMV()

Atualiza o conteúdo do parâmetro especificado no arquivo SX6, de acordo com as parametrizações informadas.

- ☒ **Sintaxe: PUTMV(cParametro, cConteudo)**
- ☒ **Parâmetros**

cParametro	Nome do parâmetro do sistema no SX6, sem a especificação da filial de sistema.
cConteudo	Conteúdo que será atribuído ao parâmetro no SX6.

SUPERGETMV()

Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Caso o parâmetro não exista será exibido um help do sistema informando a ocorrência.

Difere do GetMv() pois os parâmetros consultados são adicionados em uma área de memória, que permite que em uma nova consulta não seja necessário acessar e pesquisar o parâmetro na base de dados.

- ☒ **Sintaxe: SUPERGETMV(cParametro , IHelp , cPadrao , cFilial)**
- ☒ **Parâmetros**

cParametro	Nome do parâmetro do sistema no SX6, sem a especificação da filial de sistema.
IHelp	Se será exibida a mensagem de Help caso o parâmetro não seja encontrado no SX6.
cPadrao	Conteúdo padrão que será utilizado caso o parâmetro não exista no SX6.
cFilial	Define para qual filial será efetuada a consulta do parâmetro. Padrão → filial corrente da conexão.



Anotações

Controle de impressão

AVALIMP()

A função AVALIMP() é utilizada em relatórios específicos em substituição da função CABEC(), configurando a impressora de acordo com o driver escolhido e os parâmetros de impressão disponíveis no array aReturn, respeitando o formato utilizado pela função SETPRINT().

☑ **Sintaxe:** AVALIMP(nLimite)

☑ **Parâmetros:**

nLimite	Tamanho do relatório em colunas, podendo assumir os valores 80,132 ou 220 colunas, respectivamente para os formatos "P", "M" ou "G" de impressão.
----------------	---

☑ **Retorno:**

Caracter	String com caracteres de controle, dependente das configurações escolhidas pelo usuário e do arquivo de driver especificado.
-----------------	--

Exemplo:

```
/*/  
+-----+-----+-----+-----+-----+-----+  
| Função   | XAVALIMP   | Autor  | ARNALDO RAYMUNDO JR. | Data   | 01.01.2007 |  
+-----+-----+-----+-----+-----+-----+  
| Descrição | Exemplo de utilização da função AXCADASTRO() |  
+-----+-----+-----+-----+-----+-----+  
| Uso       | Curso ADVPL |  
+-----+-----+-----+-----+-----+-----+  
/*/  
  
USER FUNCTION XAVALIMP()  
  
LOCAL cTitulo      := PADC("AVALIMP",74)  
LOCAL cDesc1       := PADC("Demonstração do uso da função AVALIMP()",74)  
LOCAL cDesc2       := ""  
LOCAL cDesc3       := PADC("CURSO DE ADVPL",74)  
LOCAL cTamanho     := "G"  
LOCAL cLimite      := 220  
LOCAL cNatureza    := ""  
LOCAL aReturn      := {"Especial", 1,"Administração", 1, 2, 2,"",1}  
LOCAL cNomeProg    := "RAVALIMP"  
LOCAL cPerg       := PADR("RAVALIMP",10) // Compatibilização com MP10  
LOCAL nLastKey     := 0  
LOCAL cString      := "SF2"  
  
Pergunte(cPerg,.F.) // Pergunta no SX1  
  
wnrel:= SetPrint(cString,wnrel,cPerg,cTitulo,cDesc1,cDesc2,cDesc3,.T.)  
  
SetDefault(aReturn,cString)
```

Exemplo (continuação):

```
If nLastKey == 27
    Return
Endif

RptStatus({| | RunReport(cString)},cTitulo)
Return

/*
+-----+
| Função   | RUNREPORT   | Autor | ----- | Data | 01.01.2007 |
+-----+
| Descrição | Função interna de processamento utilizada pela XAVALIMP() |
+-----+
| Uso       | Curso ADVPL |
+-----+
*/

Static Function RunReport(cString)
SetPrc(0,0)

//+-----+
//| Chamada da função AVALIMP() |
//+-----+
@ 00,00 PSAY AvalImp(220)
dbSelectArea(cString)
dbSeek(xFilial()+mv_par01+mv_par03,.T.)
...

Return
```

CABEC()

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

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

☒ **Parâmetros:**

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

☒ **Parâmetros (continuação):**

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

☒ **Retorno:**

Nenhum	.
---------------	---

Exemplo:

```
#INCLUDE "protheus.ch"

/*
+-----+
| Função   | MPTR001   | Autor | ARNALDO RAYMUNDO JR. | Data | 01.01.2007 |
+-----+
| Descrição | Exemplo de utilização das funções de impressão CABEC() |
+-----+
| Uso       | Curso ADVPL
+-----+
*/

User Function MPTR001()

Local cDesc1      := "Este programa tem como objetivo imprimir relatorio "
Local cDesc2      := "de acordo com os parametros informados pelo usuario."
Local cDesc3      := "Listagem de clientes"
Local cTitulo     := "Listagem de clientes"
Local lImprime    := .T.

// Parametros da SetPrint()
Local cString      := "SA1"
Local cPerg        := ""
Local lDic         := .T. // Habilita a visualizacao do dicionario
Local aOrd         := RetSixOrd(cString)
Local lCompres     := .T. // .F. - Normal / .T. - Comprimido
Local lFilter      := .T. // Habilita o filtro para o usuario
Local cNomeProg    := "MPTR002"
Local cTamanho     := "M"
Local nTipo        := 18
Local nLimite      := 132

Default lCriaTrab := .T.

Private lEnd        := .F.
Private lAbortPrint := .F.
Private aReturn     := { "Zebrado", 1, "Administracao", 2, 2, 1, "", 1}
//aReturn[4] 1- Retrato, 2- Paisagem
//aReturn[5] 1- Em Disco, 2- Via Spool, 3- Direto na Porta, 4- Email

Private nLastKey := 0
Private m_pag     := 01
Private wnrel      := "MPTR002"

dbSelectArea("SA1")
dbSetOrder(1)
```

Exemplo (continuação):

```
//UAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//^ Monta a interface padrao com o usuario... 3
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
wnrel :=
SetPrint(cString,cNomeProg,cPerg,@cTitulo,cDesc1,cDesc2,cDesc3,lDic,aOrd,lCompre
s,cTamanho,,lFilter)

If nLastKey == 27
    Return
Endif

SetDefault(aReturn,cString,,,cTamanho,aReturn[4]) // nFormato: 1- Retrato, 2-
Paisagem

If nLastKey == 27
    Return
Endif

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

//UAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//^ Processamento. RPTSTATUS monta janela com a regra de processamento. 3
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
RptStatus({|
RunReport(cTitulo,cString,cNomeProg,cTamanho,nTipo,nLimite)},cTitulo)
Return

/*
+-----+
| Função      | RUNREPORT      | Autor | ----- | Data | 01.01.2007 |
+-----+
| Descrição   | Função interna de processamento utilizada pela MPTR001() |
+-----+
| Uso         | Curso ADVPL    |
+-----+
*/

Static Function RunReport(cTitulo,cString,cNomeProg,cTamanho,nTipo,nLimite)

Local nLin      := 80
Local cCabec1   := ""
Local cCabec2   := ""
Local cArqInd

cCabec1 := "CODIGO"+Space(2)+"LOJA"+Space(2)+"NOME REDUZIDO"+Space(9)
cCabec1 += "RAZAO SOCIAL"+Space(30)+"CNPJ"+Space(18)+"INSCR. ESTADUAL"+Space(8)
cCabec1 += "CEP"
cCabec2 := "ESTADO"+Space(2)+"MUNICIPIO"+Space(8)+"ENDERECO"

dbSelectArea("TRBSA1")
dbGoTop()

SetRequa(RecCount())
```


Exemplo (continuação):

```

While !EOF()

    If lAbortPrint .OR. nLastKey == 27
        @nLin,00 PSAY "*** CANCELADO PELO OPERADOR ***"
        Exit
    Endif

    If nLin > 55 // Salto de Página. Neste caso o formulario tem 55 linhas...
        Cabec(cTitulo,cCabec1,cCabec2,cNomeProg,cTamanho,nTipo)
        nLin := 9
    Endif

...

```

IMPCADAST()

A função IMPCADAST() cria uma interface simples que permite a impressão dos cadastros do sistema com parametrização DE/ATE.

☑ **Sintaxe:** IMPCADAST(cCab1, cCab2, cCab3, cNomeProg, cTam, nLimite, cAlias)

☑ **Parâmetros:**

cCab1	Primeira linha do cabeçalho
cCab2	Segunda linha do cabeçalho
cCab3	Terceira linha do cabeçalho
cNomeProg	Nome do programa
cTam	Tamanho do relatório nos formatos "P", "M" e "G".
nLimite	Número de colunas do relatório, seguindo o formato especificado no tamanho, aonde: "P"- 80 colunas "M"- 132 colunas "G"- 220 colunas
cAlias	Alias do arquivo de cadastro que será impresso

☑ **Retorno:**

Nenhum	.
---------------	---

MS_FLUSH()

A função MS_FLUSH() envia o spool de impressão para o dispositivo previamente especificado com a utilização das funções AVALIMP() ou SETPRINT().

☑ **Sintaxe:** MS_FLUSH()

☑ **Parâmetros:**

Nenhum	.
---------------	---

☒ **Retorno:**

Nenhum	.
--------	---

Exemplo:

```
/*/  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
| Função      | RUNREPORT      | Autor | ----- | Data | 01.01.2007 |  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
| Descrição   | Função interna de processamento utilizada pela MPTR001() |  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
| Uso         | Curso ADVPL    |  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
| Observação  | Continuação do exemplo da função CABEC() |  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
/*/  
  
Static Function RunReport(cTitulo,cString,cNomeProg,cTamanho,nTipo,nLimite)  
  
Local nLin      := 80  
Local cCabec1   := ""  
Local cCabec2   := ""  
Local cArqInd  
  
cCabec1 := "CODIGO"+Space(2)+"LOJA"+Space(2)+"NOME REDUZIDO"+Space(9)  
cCabec1 += "RAZAO SOCIAL"+Space(30)+"CNPJ"+Space(18)+"INSCR.ESTADUAL"+Space(8)  
cCabec1 += "CEP"  
cCabec2 := "ESTADO"+Space(2)+"MUNICIPIO"+Space(8)+"ENDEREÇO"  
  
dbSelectArea("TRBSA1")  
dbGoTop()  
  
SetRegua(RecCount())  
  
While !EOF()  
  
    If lAbortPrint .OR. nLastKey == 27  
        @nLin,00 PSAY "*** CANCELADO PELO OPERADOR ***"  
        Exit  
    Endif  
  
    If nLin > 55 // Salto de Página. Neste caso o formulario tem 55 linhas...  
        Cabec(cTitulo,cCabec1,cCabec2,cNomeProg,cTamanho,nTipo)  
        nLin := 9  
    Endif  
  
    // Primeira linha de detalhe:  
    @nLin,000 PSAY TRBSA1->A1_COD  
    @nLin,008 PSAY TRBSA1->A1_LOJA  
    @nLin,014 PSAY TRBSA1->A1_NREDUZ  
    @nLin,036 PSAY TRBSA1->A1_NOME  
    @nLin,078 PSAY TRBSA1->A1_CGC  
    @nLin,100 PSAY TRBSA1->A1_INSCR  
    @nLin,122 PSAY TRBSA1->A1_CEP  
    nLin++
```

Exemplo (continuação):

```
// Segunda linha de detalhe
@nLin,000 PSAY TRBSA1->A1_EST
@nLin,008 PSAY TRBSA1->A1_MUN
@nLin,025 PSAY TRBSA1->A1_END
nLin++

//Linha separadora de detalhes
@nLin,000 PSAY Replicate("-",nLimite)
nLin++

dbSkip() // Avanca o ponteiro do registro no arquivo
EndDo

SET DEVICE TO SCREEN

If aReturn[5]==1
    dbCommitAll()
    SET PRINTER TO
    OurSpool(wnrel)
Endif

MS_FLUSH()
RETURN
```

OURSPOOL()

A função OURSPOOL() executa o gerenciador de impressão da aplicação Protheus, permitindo a visualização do arquivo de impressão gerado pelo relatório no formato PostScrip® com extensão “##R”.

☒ **Sintaxe: OURSPOOL(cArquivo)**

☒ **Parâmetros:**

cArquivo	Nome do relatório a ser visualizado.
-----------------	--------------------------------------

☒ **Retorno:**

.	.
---	---

Exemplo:

```
If aReturn[5]==1 // Indica impressão em disco.
    dbCommitAll()
    SET PRINTER TO
    OurSpool(wnrel)
Endif
```

RODA()

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



Pode ser utilizado o ponto de entrada "RodaEsp" para tratamento de uma impressão específica.

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

☑ **Parâmetros:**

uPar01	Não é mais utilizado
uPar02	Não é mais utilizado
cSize	Tamanho do relatório ("P","M","G")

☑ **Retorno:**

Nenhum	.
---------------	---

Exemplo:

```
/*/  
+-----+-----+-----+-----+-----+-----+  
| Função   | TESTIMPR   | Autor | MICROSIGA           | Data | 01.01.2007 |  
+-----+-----+-----+-----+-----+-----+  
| Descrição | Exemplo de utilização da função RODA() em conjunto com a CABEC. |  
+-----+-----+-----+-----+-----+-----+  
| Uso       | Curso ADVPL |  
+-----+-----+-----+-----+-----+-----+  
/*/  
  
#include "protheus.ch"  
  
User Function TestImpr()  
Local wnrel  
Local cString := "SA1"  
Local titulo  := "Teste Impressão de Relatorios"  
Local NomeProg := "XXX"  
Local Tamanho := "M"  
  
PRIVATE aReturn := { "Zebrado", 1,"Administracao", 1, 2, 1, "",1 }  
  
wnrel:=SetPrint(cString,NomeProg,"",@titulo,"", "", "",.F.,.F.,.F.,Tamanho,,.F.)  
  
SetDefault(aReturn,cString)  
  
RptStatus({|lEnd| TestRel(@lEnd,wnRel,cString,Tamanho,NomeProg)},titulo)  
  
Return
```

Exemplo (continuação):

```
/*/  
+-----+-----+-----+-----+-----+-----+  
| Função | TESTREL | Autor | MICROSIGA | Data | 01.01.2007 |  
+-----+-----+-----+-----+-----+-----+  
| Descrição | Função interna de impressão da TestImpr(). |  
+-----+-----+-----+-----+-----+-----+  
| Uso | Curso ADVPL |  
+-----+-----+-----+-----+-----+-----+  
/*/  
User Function TestRel(lEnd,WnRel,cString,Tamanho,NomeProg)  
  
LOCAL cabec1,cabec2  
LOCAL cRodaTxt := oemtoansi("Rodapé")  
Local nCntImpr  
Local nTipo  
  
nCntImpr := 0  
li := 80  
m_pag := 1  
nTipo := 15  
titulo:= oemtoansi("Lista de Clientes")  
cabec1:= oemtoansi("COD LOJA NOME"+Space(27)+ "NOME FANTASIA")  
cabec2:=""  
  
dbSelectArea("SA1")  
dbGoTop()  
SetRegua>LastRec()  
While !Eof()  
    IncRegua()  
    If Li > 60  
        cabec(titulo,cabec1,cabec2,nomeprog,tamanho,15)  
        @ Li,0 PSAY __PrtThinLine()  
    Endif  
    nCntImpr++  
    Li++  
    @ Li,01 PSAY A1_COD  
    @ Li,05 PSAY A1_LOJA  
    @ Li,10 PSAY A1_NOME  
    @ Li,51 PSAY A1_NREDUZ  
    If Li > 60  
        Li:=66  
    Endif  
    dbSkip()  
EndDO  
  
If li != 80  
    Roda(nCntImpr,cRodaTxt,Tamanho)  
EndIf  
  
Set Device to Screen  
If aReturn[5] = 1  
    Set Printer To  
    dbCommitAll()  
    OurSpool(wnrel)  
Endif  
MS_FLUSH()  
Return
```

SETDEFAULT()

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

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

☑ **Parâmetros:**

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

☑ **Retorno:**

Nenhum	.
---------------	---

☑ **Estrutura aReturn:**

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



Anotações

SETPRC()

A função SETPRC() é utilizada para posicionar o dispositivo de impressão ativo, previamente definido pelo uso das funções AVALIMP() ou SETPRINT(), em uma linha/coluna especificada.

☑ **Sintaxe:** SETPRC(nLinha, nColuna)

☑ **Parâmetros:**

nLinha	Linha na qual deverá ser posicionado o dispositivo de impressão.
nColuna	Coluna na qual deverá ser posicionado o dispositivo de impressão.

☑ **Retorno:**

Nenhum	.
---------------	---

Exemplo:

```
aReturn := { "", 1, "", 2, 3, cPorta, "", IndexOrd() }
SetPrint(Alias(), "", "", "", F., EPSON.DRV, T., cPorta)
if nLastKey == 27
Return (.F.)
Endif
SetDefault(aReturn, Alias())
SetPrc(0,0)
```

SETPRINT()

A função SetPrint() cria uma interface padrão onde as opções de impressão de um relatório podem ser configuradas. Basicamente duas variáveis m_pag e aReturn precisam ser declaradas como privadas (private) antes de executar a SetPrint(), sendo que:

- ☐ **m_pag**: controla o número de páginas.
- ☐ **aReturn**: vetor contendo as opções de impressão, sendo sua estrutura básica composta de 8 (oito) elementos.

Após confirmada, os dados são armazenados no vetor aReturn que será passado como parâmetro para função SetDefault().

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

☑ **Parâmetros:**

cAlias	Alias do arquivo a ser impresso.
cProgram	Nome do arquivo a ser gerado em disco.
cPergunte	Grupo de perguntas cadastrado no dicionário SX1.
cTitle	Título do relatório.

☒ **Parâmetros (continuação):**

cDesc1	Descrição do relatório.
cDesc2	Continuação da descrição do relatório.
cDesc3	Continuação da descrição do relatório.
IDic	Utilizado na impressão de cadastro genérico permite a escolha dos campos a serem impressos. Se o parametro cAlias não for informado o valor padrão assumido será .F.
aOrd	Ordem(s) de impressão.
ICompres	Se verdadeiro (.T.) permite escolher o formato da impressão, o valor padrão assumido será .T.
cSize	Tamanho do relatório "P", "M" ou "G".
uParm12	Parâmetro reservado
IFilter	Se verdadeiro (.T.) permite a utilização do assistente de filtro, o valor padrão assumido será .T.
ICrystal	Se verdadeiro (.T.) permite integração com Crystal Report, o valor padrão assumido será .F.
cNameDrv	Nome de um driver de impressão.
uParm16	Parâmetro reservado.
IServer	Se verdadeiro (.T.) força impressão no servidor.
cPortPrint	Define uma porta de impressão padrão.

☒ **Retorno:**

Caracter	Nome do Relatório
-----------------	-------------------

☒ **Estrutura aReturn:**

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

Controle de processamentos

ABREEXCL()

A função ABREEXCL() fecha o arquivo cujo alias está expresso em <cAlias> e o reabre em modo exclusivo para proceder operações em que isto é necessário, como por exemplo, PACK. Se outra estação estiver usando o arquivo, o retorno será .F..

☑ **Sintaxe:** ABREEXCL(cAlias)

☑ **Parâmetros:**

cAlias	Alias do arquivo que será re-aberto em modo exclusivo.
---------------	--

☑ **Retorno:**

Lógico	Indica se foi possível abrir o arquivo em modo exclusivo.
---------------	---

CLOSEOPEN()

A função CLOSEOPEN() é utilizada para fechar e re-abrir uma lista de arquivos especificada.

☑ **Sintaxe:** CLOSEOPEN(aClose, aOpen)

☑ **Parâmetros:**

aClose	Array contendo os Aliases dos arquivos que deverão ser fechados.
aOpen	Array contendo os Aliases dos arquivos que deverão ser abertos.

☑ **Retorno:**

Lógico	Indica se todos os arquivos especificados em aOpen foram abertos com sucesso.
---------------	---

CLOSESFILE()

A função CLOSESFILE() fecha todos os arquivos em uso pela conexão, com exceção dos SXs (inclusive SIX), SM2 e SM4.

☑ **Sintaxe:** CLOSESFILE(cAlias)

☑ **Parâmetros:**

cAlias	String contendo os nomes dos demais Aliases que não deverão ser fechados, separando os itens com "/".
---------------	---

☑ **Retorno:**

Lógico	Indica se todos os arquivos foram fechados com sucesso.
---------------	---

CHKFILE()

A função CHKFILE() retorna verdadeiro (.T.) se o arquivo já estiver aberto ou se o Alias não for informado. Sempre que desejar mudar o modo de acesso do arquivo (de compartilhado para exclusivo ou vice-versa), feche-o antes de chamá-la.

☒ **Sintaxe:** ChkFile(cAlias,IExcl,cNewAlias)

☒ **Parâmetros:**

cAlias	Alias do arquivo a ser re-aberto.
IExcl	Se for informado verdadeiro (.T.), o arquivo será aberto em modo exclusivo, caso contrário, o arquivo será aberto em modo compartilhado. Se este parâmetro não for informado, será assumido falso (.F.).
cNewAlias	Novo Alias para re-abertura do arquivo.

☒ **Retorno:**

Lógico	Indica se o arquivo foi re-aberto com sucesso.
---------------	--

Exemplo:

```
dbSelectArea("SA1")
dbCloseArea()
IOk := .T.
While .T.
    IF !ChkFile("SA1",.T.)
        nResp := Alert("Outro usuário usando! Tenta de novo?",{ "Sim","Nao"})
        If nResp == 2
            IOk := .F.
            Exit
        Endif
    Endif
EndDo
If IOk
    // Faz o processamento com o arquivo...
Endif
// Finaliza
If Select("SA1")
    dbCloseArea()
Endif
ChkFile("SA1",.F.)
Return
```



Anotações

CONOUT()

A função CONOUT() permite a exibição de uma mensagem de texto no console do Server do Protheus. Caso o Protheus esteja configurado como serviço a mensagem será gravada no arquivo de log.

☑ **Sintaxe:** CONOUT(cMensagem)

☑ **Parâmetros:**

cMensagem	String contendo a mensagem que deverá ser exibida no console do Protheus.
------------------	---

☑ **Retorno:**

Nenhum	.
---------------	---

CRIAVAR()

A função CRIAVAR() cria uma variável, retornando o valor do campo, de acordo com o dicionário de dados, inclusive avaliando o inicializador padrão, permitindo um retorno de acordo com o tipo de dado definido no dicionário.

☑ **Sintaxe:** CriaVar(cCampo,II niPad,cLado)

☑ **Parâmetros:**

cCampo	Nome do campo
II niPad	Indica se considera (.T.) ou não (.F.) o inicializador
cLado	Se a variável for caracter, cLado pode ser: "C" - centralizado, "L" - esquerdo ou "R" - direito.

☑ **Retorno:**

Indefinido	Tipo de dado de acordo com o dicionário de dados, considerando inicializador padrão
-------------------	---

Exemplo:

```
// Exemplo do uso da função CriaVar:  
cNumNota := CriaVar("F2_DOC") // Retorna o conteúdo do  
// inicializador padrão,  
// se existir, ou espaços em branco  
Alert(cNumNota)  
Return
```

DISARMTRANSACTION()

A função DISARMTRANSACTION() é utilizada para realizar um "RollBack" de uma transação aberta com o comando BEGIN TRANSACTION e delimitada com o comando END TRANSACTION.

Ao utilizar esta função, todas as alterações realizadas no intervalo delimitado pela transação são desfeitas, restaurando a situação da base de dados ao ponto imediatamente anterior ao início do processamento.



Importante

O uso da função DISARMTRANSACTION() não finaliza a conexão ou o processamento corrente.

Caso seja necessário além de desfazer as alterações, finalizar o processamento, deverá ser utilizada a função USEREXCEPTION().

☒ **Sintaxe:** DISARMTRANSACTION()

☒ **Parâmetros:**

Nenhum

.

☒ **Retorno:**

Nenhum

.

Exemplo:

```
IMsErroAuto := .F.  
MSExecAuto({|x,y| MATA240(x,y)}, aCampos, 3)
```

```
If IMsErroAuto
```

```
    aAutoErro := GETAUTOGRLOG()  
    DisarmTransaction()  
    MostraErro()
```

```
EndIf
```



Anotações

EXECBLOCK()

A função EXECBLOCK() executa uma função de usuário que esteja compilada no repositório. Esta função é normalmente utilizada pelas rotinas padrões da aplicação Protheus para executar pontos de entrada durante seu processamento.



A função de usuário executada através da EXECBLOCK() não recebe parâmetros diretamente, sendo que estes estarão disponíveis em uma variável private denominada **PARAMIXB**.



A variável **PARAMIXB** é o reflexo do parâmetro xParam, definido na execução da função EXECBLOCK(). Caso seja necessária a passagem de várias informações, as mesmas deverão ser definidas na forma de um array, tornando **PARAMIXB** um array também, a ser tratado na função de usuário que será executada.

EXISTBLOCK()

A função EXISTBLOCK() verifica a existência de uma função de usuário compilada no repositório de objetos da aplicação Protheus. Esta função é normalmente utilizada nas rotinas padrões da aplicação Protheus para determinar a existência de um ponto de entrada e permitir sua posterior execução.

☑ **Sintaxe:** EXISTBLOCK(cFunção)

☑ **Parâmetros:**

cFunção	Nome da função que será avaliada.
----------------	-----------------------------------

☑ **Retorno:**

Lógico	Indica se a função de usuário existe compilada no repositório de objetos corrente.
---------------	--

Exemplo:

```
IF EXISTBLOCK("MT100GRV")
    EXECBLOCK("MT100GRV",.F.,.F.,aParam)
ENDIF
```

☑ **Sintaxe:** EXECBLOCK(cFunção, IReserv1, IReserv2, xParam)

☑ **Parâmetros:**

cFunção	Nome da função de usuário que será executada.
IReserv1	Parâmetro de uso reservado da aplicação. Definir como .F.
IReserv2	Parâmetro de uso reservado da aplicação. Definir como .F.
xParam	Conteúdo que ficará disponível na função de usuário executada, na forma da variável private PARAMIXB .

☑ **Retorno:**

Indefinido	O retorno da EXECBLOCK() é definido pela função que será executada.
-------------------	---

Exemplo:

```
aParam := {cNota, cSerie, cFornece, cLoja}

IF EXISTBLOCK("MT100GRV")
    lGravou := EXECBLOCK("MT100GRV",.F.,.F.,aParam)
ENDIF

USER FUNCTION MT100GRV()

LOCAL cNota := PARAMIXB[1]
LOCAL cSerie:= PARAMIXB[1]
LOCAL cFornece:= PARAMIXB[1]
LOCAL cLoja:= PARAMIXB[1]

RETURN .T.
```

ERRORBLOCK()

A função ERRORBLOCK() efetua o tratamento de erros e define a atuação de um handler de erros sempre que ocorrer um erro em tempo de execução. O manipulador de erros é especificado como um bloco de código da seguinte forma:

☐ { |<objError>| <lista de expressões>, ... }, onde:

<objError> é um error object que contém informações sobre o erro. Dentro do bloco de código, podem ser enviadas mensagens ao error object para obter informações sobre o erro. Se o bloco de tratamento de erros retornar verdadeiro (.T.), a operação que falhou é repetida, e se retornar falso (.F.), o processamento recomeça.

Se não foi especificado nenhum <bErrorHandler> utilizando ERRORBLOCK() e ocorrer um erro em tempo de execução, o bloco de tratamento ao de erros padrão é avaliado. Este manipulador de erros exibe uma mensagem descritiva na tela, ajusta a função ERRORLEVEL() para 1, e depois sai do programa (QUIT).

Como ERRORBLOCK() retorna o bloco de tratamento ao de erros corrente, é possível especificar um bloco de tratamento de erros para uma operação gravando-se o bloco de manipulação de erros corrente e depois recuperando-o após o final da operação. Além disso,

uma importante consequência do fato de os blocos de tratamento de erros serem especificados como blocos de código, é que eles podem ser passados para rotinas e funções definidas por usuário e depois retornadas como valores.

- ☑ **Sintaxe: ERRORBLOCK (< bErrorHandler >)**
- ☑ **Parâmetros:**

bErrorHandler	O bloco de código a ser executado toda vez que ocorrer um erro em tempo de execução. Quando avaliado, o <bErrorHandler> é passado na forma de um objeto erro como um argumento pelo sistema.
----------------------	--

- ☑ **Retorno:**

Code-block	Retorna o bloco de código corrente que tratar o erro. Caso não tenha sido enviado nenhum bloco de tratamento de erro desde que o programa foi invocado, ERRORBLOCK() retorna o bloco de tratamento de erro padrão.
-------------------	--

Exemplo:

```
Function CA010Form()
LOCAL xResult
LOCAL cForm:= Upper(&(ReadVar()))
LOCAL bBlock:= ErrorBlock( { |e| ChecErro(e) } )
LOCAL cOutMod
Local lOptimize := GetNewPar("MV_OPTNFE",.F.) .Or. GetNewPar("MV_OPTNFS",.F.)

PRIVATE lRet:=.T.

cVarOutMod := If(Type("cVarOutMod") = "U", "", cVarOutMod)
cOutMod     := cVarOutMod + If(Right(cVarOutMod, 1) = ",", "", ",")

While ! Empty(cOutMod)
    If Left(cOutMod, At(",", cOutMod) - 1) $ Upper(cForm)           // no modulo
        Help( " ",1,"ERR_FORM","Variavel nao disponivel para este modulo"
        Return .F.
    Endif
    cOutMod := Subs(cOutMod, At(",", cOutMod) + 1)
EndDo
If ("LERSTR"$cForm .or. "LERVAL"$cForm .or. "LERDATA"$cForm) .And. M->I5_CODIGO >
"499"
    Help( " ",1,"CA010TXT")
    ErrorBlock(bBlock)
    Return .F.
Endif
BEGIN SEQUENCE
    If !"EXECBLOCK"$cForm .and. !"LERSTR"$cForm .And.; // nao executa execblock
        !"LERVAL"$cForm .And.; // nem funcao de leitura
        !"LERDATA"$cForm .And.; // de texto no cadastramento
        IIf(!lOptimize,.T.,!"CTBANFS"$cForm .And. !"CTBANFE"$cForm)
        xResult := &cForm
    Endif
END SEQUENCE
ErrorBlock(bBlock)
Return lRet
```

FINAL()

A função FINAL() executa as operações básicas que garantem a integridade dos dados ao finalizar o sistema desmontando as transações (se houver), desbloqueando os semáforos e fechando as tabelas abertas, finalizando-o em seguida.

☑ **Sintaxe:** Final([cMensagem1], [cMensagem2])

☑ **Parâmetros:**

cMensagem1	Primeira mensagem
cMensagem2	Segunda mensagem

☑ **Retorno:**

Nenhum	.
---------------	---

Exemplo:

```
User Function ValidUser( cUsuario, cSenha )  
  
Local cMensag1 := "Usuário inválido!"  
Local cMensag2 := "Opção disponível para usuários Administradores!"  
  
If !PswAdmin( cUsuario, cSenha )  
    Final( cMensag1, cMensag2 )  
EndIf  
  
Return
```

FINDFUNCTION()

A função FINDFUNCTION() tem como objetivo verificar se uma determinada função se encontra no repositório de objetos e até mesmo do binário do Protheus, sendo uma função básica da linguagem.

☑ **Sintaxe:** FINDFUNCTION(cFunção)

☑ **Parâmetros:**

cFunção	Nome da função que será avaliada no repositório de objetos corrente.
----------------	--

☑ **Retorno:**

Lógico	Indica se a função existe compilada no repositório de objetos corrente.
---------------	---

FUNDESC()

A função FunDesc() retornará a descrição de uma opção selecionada no menu da aplicação.

☒ **Sintaxe:** FUNDESC()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Caracter	Descrição da opção selecionada no menu da aplicação.
----------	--

FUNNAME()

A função FunName() retornará o nome de uma função executada a partir de um menu da aplicação.

☒ **Sintaxe:** FUNNAME()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Caracter	Nome da função executada a partir do menu da aplicação.
----------	---

GETAREA()

Função utilizada para proteger o ambiente ativo no momento de algum processamento específico. Para salvar uma outra área de trabalho (alias) que não o ativo, a função GetArea() deve ser executada dentro do alias: ALIAS->(GetArea()).

☒ **Sintaxe:** GETAREA()

☒ **Retorno:** Array contendo {Alias(),IndexOrd(),Recno()}

☒ **Parâmetros**

Nenhum	.
--------	---

GETCOUNTRYLIST()

A função GETCOUNTRYLIST() retorna um array de duas dimensões contendo informações dos países localizados.

☑ **Sintaxe:** GetCountryList()

☑ **Parâmetros:**

Nenhum	.
--------	---

☑ **Retorno:**

Array	Array de duas dimensões, sendo uma linha para cada país localizado, contendo em cada posição a sigla dos países, o nome do país e a identificação do país com dois dígitos.
-------	---

Exemplo:

```
Local aArray := GetCountryList()
Local cSigla := GetMv( "MV_PAISLOC" )
Local nPos

nPos := Ascan( aArray, { |d| d[1] == Upper(cSigla) } )
If nPos > 0
    APMsgInfo( "País de localização " + aArray[nPos,2] )
EndIf
```

ISINCALLSTACK()

A função ISINCALLSTACK() verifica se uma determinada função está existe dentro da pilha de chamadas do processamento corrente.

☑ **Sintaxe:** IsInCallStack(cIsInCallStack , cStackExit)

☑ **Parâmetros:**

cIsInCallStack	Nome da função que desejasse pesquisar na pilha.
cStackExit	String que identifica o ponto em que desejasse finalizar a busca. Caso não seja informada, será utilizada como padrão a expressão "STACK_EXIT".

☑ **Retorno:**

Lógico	Indica se a função especificada encontrasse na pilha de chamadas do processamento corrente, até o ponto de saída especificado.
--------	--

REGTOMEMORY()

Inicializa as variáveis de memória identificadas pelo uso do alias "M->" de acordo com a estrutura e/ou informações contidas no arquivo definido como referência.

☒ **Sintaxe:** REGTOMEMORY(cAlias, IlInclui)

☒ **Parâmetros:**

cAlias	Alias do arquivo que será utilizado como referência para inicialização das variáveis de memória.
IlInclui	Identifica se as variáveis deverão ser inicializadas com conteúdos padrões, ou contendo as informações do registro posicionado do alias especificado.

☒ **Retorno:**

Nenhum	.
---------------	---



Anotações

RESTAREA()

Função utilizada para devolver a situação do ambiente salva através do comando GETAREA(). Deve-se observar que a última área restaurada é a área que ficará ativa para a aplicação.

- ☒ **Sintaxe: RESTAREA(aArea)**
- ☒ **Parâmetros**

aArea	Array contendo: {cAlias, nOrdem, nRecno}, normalmente gerado pelo uso da função GetArea().
--------------	--

Exemplo:

```
// ALIAS ATIVO ANTES DA EXECUÇÃO DA ROTINA → SN3
User Function XATF001()

LOCAL cVar
LOCAL aArea := GetArea()
LOCAL IRet := .T.

cVar := &(amp;ReadVar())

dbSelectArea("SX5")
IF !dbSeek(xFilial()+"Z1"+cVar)

    cSTR0001 := "REAV - Tipo de Reavaliacao"
    cSTR0002 := "Informe um tipo de reavaliacao valido"
    cSTR0003 := "Continuar"
    Aviso(cSTR0001,cSTR0002,{cSTR0003},2)
    IRet := .F.

ENDIF

RestArea(aArea)
Return( IRet )
```

USEREXCEPTION()

A função USEREXCEPTION() tem o objetivo de forçar um erro em ADVPL de forma que possamos tratar de alguma forma. USEREXCEPTION() recebe uma string contendo uma descrição do erro, essa descrição será exibida de acordo com o ambiente que se está executando, caso um ambiente ERP, será exibida uma tela de erro.

- ☒ **Sintaxe: USEREXCEPTION(cMensagem)**
- ☒ **Parâmetros:**

cMensagem	Mensagem que será exibida no cabeçalho do erro, contendo a explicação da exceção.
------------------	---

- ☒ **Retorno:**

Nenhum	.
--------	---



Anotações

Utilização de recursos do ambiente ERP

AJUSTASX1()

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

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

☑ **Parâmetros:**

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

☑ **Retorno:**

Nenhum	.
---------------	---

☑ **Estrutura – Item do array aPerg:**

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

☑ Estrutura – Item do array aPerg (continuação):

23	X1_DEFSPA3	Caractere	Descrição da opção X do combo em espanhol
24	X1_DEFENG3	Caractere	Descrição da opção X do combo em inglês
25	X1_CNT03	Caractere	Não é informado
26	X1_VAR04	Caractere	Não é informado
27	X1_DEF04	Caractere	Descrição da opção X do combo em português
28	X1_DEFSPA4	Caractere	Descrição da opção X do combo em espanhol
29	X1_DEFENG4	Caractere	Descrição da opção X do combo em inglês
30	X1_CNT04	Caractere	Não é informado
31	X1_VAR05	Caractere	Não é informado
32	X1_DEF05	Caractere	Descrição da opção X do combo em português
33	X1_DEFSPA5	Caractere	Descrição da opção X do combo em espanhol
34	X1_DEFENG5	Caractere	Descrição da opção X do combo em inglês
35	X1_CNT05	Caractere	Não é informado
36	X1_F3	Caractere	Código da consulta F3 vinculada ao parâmetro
37	X1_GRPSXG	Caractere	Código do grupo de campos SXG para atualização automática, quando o grupo for alterado.
38	X1_PYME	Caractere	Se a pergunta estará disponível no ambiente Pyme
39	X1_HELP	Caractere	Conteúdo do campo X1_HELP
40	X1_PICTURE	Caractere	Picture de formatação do conteúdo do campo.
41	aHelpPor	Array	Vetor simples contendo as linhas de help em português para o parâmetro. Trabalhar com linhas de até 40 caracteres.
42	aHelpEng	Array	Vetor simples contendo as linhas de help em inglês para o parâmetro. Trabalhar com linhas de até 40 caracteres.
43	aHelpSpa	Array	Vetor simples contendo as linhas de help em espanhol para o parâmetro. Trabalhar com linhas de até 40 caracteres.



Anotações

ALLUSERS()

A função ALLUSERS() retorna um array multidimensional contendo as informações dos usuários do sistema.

☑ **Sintaxe:** ALLUSERS()

☑ **Parâmetros:**

Nenhum

.

☑ **Retorno:**

Array

Array multidimensional contendo as informações dos usuários do sistema, aonde para cada usuário serão demonstradas as seguintes informações:

aArray[x][1] → Configurações gerais de acesso

aArray[x][2] → Configurações de impressão

aArray[x][3] → Configurações de acesso aos módulos

☑ **Array de informações dos usuários: Configurações gerais de acesso**

Elemento	Descrição	Tipo	Qtd.
1			
1	ID	C	6
2	Nome	C	15
3	Senha	C	6
4	Nome Completo	C	30
5	Vetor com nº últimas senhas	A	--
6	Data de validade	D	8
7	Quantas vezes para expirar	N	4
8	Autorizado a alterar a senha	L	1
9	Alterar a senha no próximo logon	L	1
10	Vetor com os grupos	A	--
11	ID do superior	C	6
12	Departamento	C	30
13	Cargo	C	30
14	E-Mail	C	130
15	Número de acessos simultâneos	N	4
16	Data da última alteração	D	8
17	Usuário bloqueado	L	1
18	Número de dígitos para o ano	N	1
19	Listner de ligações	L	1
20	Ramal	C	4

☒ **Array de informações dos usuários: Configurações de impressão**

Elemento	Descrição	Tipo	Qtd.
2			
1	Vetor com horários de acesso	A	--
2	Idioma	N	1
3	Diretório	C	100
4	Impressora	C	--
5	Acessos	C	512
6	Vetor com empresas	A	--
7	Ponto de entrada	C	10
8	Tipo de impressão	N	1
9	Formato	N	1
10	Ambiente	N	1
11	Prioridade p/ config. do grupo	L	1
12	Opção de impressão	C	50
13	Acesso a outros dir de impressão	L	1

☒ **Array de informações dos usuários: Configurações de acesso aos módulos**

Elemento	Descrição	Tipo	Qtd.
3			
1	Módulo+nível+menu	C	



Anotações

ALLGROUPS()

A função ALLGROUPS() retorna um array multidimensional contendo as informações dos grupos de usuários do sistema.

☑ **Sintaxe: ALLGROUPS()**

☑ **Parâmetros:**

Nenhum	.
--------	---

☑ **Retorno:**

Array	Array multidimensional contendo as informações dos grupos de usuários do sistema, aonde para cada grupo serão demonstradas as seguintes informações: aArray[x][1] → Configurações gerais de acesso aArray[x][2] → Configurações de acesso aos módulos
--------------	---

☑ **Array de informações dos grupos: Configurações gerais de acesso**

Elemento	Descrição	Tipo	Qtd.
1			
1	ID	C	6
2	Nome	C	20
3	Vetor com horários de acesso	A	
4	Data de validade	D	8
5	Quantas vezes para expirar	N	4
6	Autorizado a alterar a senha	L	1
7	Idioma	N	1
8	Diretório	C	100
9	Impressora	C	
10	Acessos	C	512
11	Vetor com empresas	A	
12	Data da última alteração	D	8
13	Tipo de impressão	N	1
14	Formato	N	1
15	Ambiente	N	1
16	Opção de impressão	L	1
17	Acesso a outros Dir de impressão	L	1

☑ **Array de informações dos grupos: Configurações de acesso aos módulos**

Elemento	Descrição	Tipo	Qtd.
2			
1	Modulo+nível+menu	C	

CGC()

A função CGC() valida o CGC digitado, utilizando o algoritmo nacional para verificação do dígito de controle.

☒ **Sintaxe:** CGC(cCGC)

☒ **Parâmetros:**

cCGC	String contendo o CGC a ser validado
-------------	--------------------------------------

☒ **Retorno:**

Lógico	Indica se o CGC informado é válido.
---------------	-------------------------------------

CONPAD1()

A função CONPAD1() exibe uma tela de consulta padrão baseada no Dicionário de Dados (SXB).

☒ **Sintaxe:** ConPad1 ([uPar1] , [uPar2] , [uPar3] , cAlias , [cCampoRet] , [uPar4] , [IOnlyView])

☒ **Parâmetros:**

uPar	Parâmetro reservado.
uPar2	Parâmetro reservado.
uPar3	Parâmetro reservado.
cAlias	Consulta padrão cadastrada no Dicionário de Dados (SXB) a ser utilizada.
cCampoRet	Nome da variável ou campo que receberá o retorno da consulta padrão.
uPar4	Parâmetro Reservado.
IOnlyView	Indica se será somente para visualização.

☒ **Retorno:**

Nenhum	.
---------------	---



Anotações

DATAVALIDA()

A função DATAVALIDA() retorna a primeira data válida a partir de uma data especificada como referência, considerando inclusive a data informada para análise.

☑ **Sintaxe: DATAVALIDA(dData)**

☑ **Parâmetros:**

dData	Data a partir da qual será avaliada a próxima data válida, considerando-a inclusive como uma possibilidade.
--------------	---

☑ **Retorno:**

Data	Próxima data válida, desconsiderando sábados, domingos e os feriados cadastrados no sistema.
-------------	--

EXISTINI()

A função EXISTINI() verifica se o campo possui inicializador padrão.

☑ **Sintaxe: EXISTINI(cCampo)**

☑ **Parâmetros:**

cCampo	Nome do campo para verificação.
---------------	---------------------------------

☑ **Retorno:**

Lógico	Indica se o campo possui um inicializador padrão.
---------------	---

Exemplo:

```
// Exemplo de uso da funcao ExistIni:  
// Se existir inicializador no campo B1_COD:  
If ExistIni("B1_COD")  
    // Executa o inicializador:  
    cCod := CriaVar("B1_COD")  
Endif  
  
Return
```

EXTENSO()

A função EXTENSO() retorna uma string referente a descrição por extenso de um valor numérico, sendo comumente utilizada para impressão de cheques, valor de duplicatas, etc.

☑ **Sintaxe: Extenso(nValor, lQtd, nMoeda)**

☑ **Parâmetros:**

nValor	Valor para geração do extenso.
lQtd	Indica se o valor representa uma quantidade (.T.) ou dinheiro (.F.)
nMoeda	Para qual moeda do sistema deve ser o extenso.

☑ **Retorno:**

String	Descrição do valor por extenso.
---------------	---------------------------------

FORMULA()

Interpreta uma fórmula cadastrada. Esta função interpreta uma fórmula, previamente cadastrada no Arquivo SM4 através do Módulo Configurador, e retorna o resultado com tipo de dado de acordo com a própria fórmula.

☑ **Sintaxe: Formula(cFormula)**

☑ **Parâmetros:**

cFormula	Código da fórmula a ser avaliada e cadastrada no SM4 – Cadastro de Fórmulas.
-----------------	--

☑ **Retorno:**

Indefinido	Resultado da interpretação da fórmula cadastrada no SM4.
-------------------	--

GETADVVAL()

A função GETADVVAL() executa uma pesquisa em um arquivo pela chave de busca e na ordem especificadas, possibilitando o retorno de um ou mais campos.

☑ **Sintaxe: GetAdvFVal(cAlias,uCpo,uChave,nOrder,uDef)**

☑ **Parâmetros:**

cAlias	Alias do arquivo
uCpo	Nome de um campo ou array contendo os nomes dos campos desejados
uChave	Chave para a pesquisa
nOrder	Ordem do índice para a pesquisa
uDef	Valor ou array "default" para ser retornado caso a chave não seja encontrada

☒ **Retorno:**

Indefinido	Retorna o conteúdo de um campo ou array com o conteúdo de vários campos
-------------------	---



Importante

A função GETADVVAL() difere da função POSICIONE() apenas por permitir o retorno de vários campos em uma única consulta.

As duas funções devem ser protegidas por GETAREA() / RESTAREA() dependendo da aplicação.

HELP()

Esta função exibe a ajuda especificada para o campo e permite sua edição. Se for um help novo, escreve-se o texto em tempo de execução.

☒ **Sintaxe: Help(cHelp,nLinha, cTitulo, uPar4,cMensagem,nLinMen,nColMen)**

☒ **Parâmetros:**

cHelp	Nome da Rotina chamadora do help. (sempre branco)
nLinha	Número da linha da rotina chamadora. (sempre 1)
cTitulo	Título do help
uPar4	Sempre NIL
cMensagem	Mensagem a ser exibida para o Help.
nLinMen	Número de linhas da Mensagem. (relativa à janela)
nColMen	Número de colunas da Mensagem. (relativa à janela)

☒ **Retorno:**

Nenhum	.
---------------	---



Importante

A função HELP() é tratada na execução das rotinas com o recurso de MSEXCAUTO(), permitindo a captura e exibição da mensagem no log de processamento.

Por esta razão, em rotinas que podem ser chamadas através da função MSEXCAUTO() deve-se sempre utilizar avisos utilizando esta função, para que este tipo de processamento não seja travado indevidamente.

Exemplo:

```
IF !Auto // Se for rotina automática
    Help("ESPECIFICO",1,"HELP","PROCESSAMENTO","Parâmetros do JOB Inválidos",1,0)
ELSE
    MsgAlert("Parâmetros do JOB Inválidos", "PROCESSAMENTO")
ENDIF
```

MESEXTENSO()

A função MESEXTENSO() retorna o nome de um mês por extenso.

☑ **Sintaxe:** MESEXTENSO(nMes)

☑ **Parâmetros:**

nMes	Indica o número do mês a ter seu nome escrito por extenso.
-------------	--



Importante

Este parâmetro pode ser definido também como caracter ou como data.

☑ **Retorno:**

String	Nome do mês indicado por extenso.
---------------	-----------------------------------

OBRIGATORIO()

A função OBRIGATORIO() avalia se todos os campos obrigatórios de uma Enchoice() foram digitados.

☑ **Sintaxe:** OBRIGATORIO(aGets, aTela, aTitulos)

☑ **Parâmetros:**

aGets	Variável PRIVATE tratada pelo objeto Enchoice(), previamente definida no fonte.
aTela	Variável PRIVATE tratada pelo objeto Enchoice(), previamente definida no fonte.
aTitulos	Array contendo os títulos dos campos exibidos na Enchoice().

☑ **Retorno:**

Lógico	Indica se todos os campos obrigatórios foram preenchidos.
---------------	---



Anotações

Exemplo:

```
#INCLUDE "protheus.ch"

/*
+-----+
| Programa | ATFA010A | Autor | ARNALDO R. JUNIOR | Data |
+-----+
| Desc.    | Cadastro de dados complementares do bem - Ativo Fixo |
+-----+
| Uso      | Curso de ADVPL |
+-----+
*/

User Function ATFA010A()

Private cCadastro := "Atualizacao de dados do bem"
Private aRotina := { {"Pesquisar" , "AxPesqui" , 0,1} , ;
                    {"Visualizar" , "AxVisual" , 0,2} , ;
                    {"Atualizar" , "U_A010AATU" , 0,4} }

Private cDelFunc := ".T."
Private cString := "SN1"

dbSelectArea("SN1")
dbSetOrder(1)
dbSelectArea(cString)
mBrowse( 6,1,22,75,cString)

Return

/*
+-----+
| Programa | A010AATU | Autor | ARNALDO R. JUNIOR | Data |
+-----+
| Desc.    | Atualização de dados do bem - Ativo Fixo |
+-----+
| Uso      | Curso de ADVPL |
+-----+
*/

User Function A010AATU(cAlias,nReg,nOpc)

Local aCpoEnch      := {}
Local aAlter        := {}

Local aButtons      := {}
Local cAliasE        := cAlias
Local aAlterEnch     := {}
Local aPos           := {015,000,400,600}
Local nModelo        :=
Local lF3            := .F.
Local lMemoria        := .T.
Local lColumn        := .F.
Local caTela         := ""
Local lNoFolder      := .F.
Local lProperty      := .F.
```


Exemplo (continuação):

```
Private oDlg
Private oGetD
Private oEnch
Private aTELA[0][0] // Variáveis que serão atualizadas pela Enchoice()
Private aGETS[0] // e utilizadas pela função OBRIGATORIO()

DbSelectArea("SX3")
DbSetOrder(1)
DbSeek(cAliasE)
//+-----+
//|Campos da enchoice |
//+-----+
While !Eof().And. SX3->X3_ARQUIVO == cAliasE
    If !(SX3->X3_CAMPO $ "A1_FILIAL").And. cNivel >= SX3->X3_NIVEL .And.
X3Uso(SX3->X3_USADO)
        AAdd(aCpoEnch,SX3->X3_CAMPO)
    EndIf
    DbSkip()
End
//+-----+
//|Campos alteráveis da enchoice |
//+-----+
AADD(aAlterEnch,"N1_TIPOADT") // Controle de Adiantamentos
AADD(aAlterEnch,"N1_DESCRIC") // Descrição
AADD(aAlterEnch,"N1_CHAPA") // Numero da plaqueta
AADD(aAlterEnch,"N1_FORNEC") // Fornecedor
AADD(aAlterEnch,"N1_LOJA") // Loja do Fornecedor
AADD(aAlterEnch,"N1_NSERIE") // Serie da Nota
AADD(aAlterEnch,"N1_NFISCAL") // Numero da Nota
AADD(aAlterEnch,"N1_NFITEM") // Item da Nota
AADD(aAlterEnch,"N1_UM") // Unidade de Medida
AADD(aAlterEnch,"N1_PRODUTO") // Código do Produto
AADD(aAlterEnch,"N1_PEDIDO") // Codigo do Pedido de Compras
AADD(aAlterEnch,"N1_ITEMPED") // Item do Pedido de Compras
AADD(aAlterEnch,"N1_PRCIMP") // Codigo do Processo de Importacao
AADD(aAlterEnch,"N1_CODPAIS") // Codigo do Pais
AADD(aAlterEnch,"N1_ORIGCPR") // Origem de Compras
AADD(aAlterEnch,"N1_CODSP") // Codigo da SP Interna
AADD(aAlterEnch,"N1_CHASSIS") // Numero de serie

//+-----+
//|Montagem do DIALOG |
//+-----+
DEFINE MSDIALOG oDlg TITLE cCadastro FROM 000,000 TO 400,600 PIXEL
    RegToMemory("SN1", .F.)

    oEnch := MsMGet():New(cAliasE, nReg, nOpc, /*aCRA*/, /*cLetra*/,;
/*cTexto*/, aCpoEnch,aPos,aAlterEnch, nModelo, /*nColMens*/,;
/*cMensagem*/, /*cTudoOk*/, oDlg, lF3, lMemoria, lColumn,;
caTela, lNoFolder, lProperty)

ACTIVATE MSDIALOG oDlg CENTERED;
ON INIT EnchoiceBar(oDlg, {|| IIF(A010AGRV(aCpoEnch,aAlterEnch,nOpc),;
oDlg:End(),.F.)},; // Botão OK
{||oDlg:End()},,aButtons) // Botão Cancelar

RETURN
```

Exemplo (continuação):

```
/*/  
+-----+  
| Programa | A010AGRV | Autor | ARNALDO R. JUNIOR | Data | |  
+-----+  
| Desc. | Validação da enchoice e gravação dos dados do bem | |  
+-----+  
| Uso | Curso de ADVPL | |  
+-----+  
/*/  
Static Function A010AGRV(aCpos,aAlter,nOpc)  
  
Local aArea := GetArea()  
Local nX := 0  
  
IF !Obrigatorio(aGets,aTela) /*Valida o cabeçalho*/  
Return .F.  
ENDIF  
  
// Atualizacao dos campos passíveis de alteracao no SN1  
RecLock("SN1",.F.)  
For nX := 1 to Len(aAlter)  
SN1->&(aAlter[nX]) := M->&(aAlter[nX])  
Next nX  
MsUnLock()  
  
Return .T.
```

OPENFILE()

A função OPENFILE() exibe o diagnóstico de arquivos, verificando a existência dos arquivos de dados e os índices do sistema, criando caso não existam e abre os arquivos de acordo com o módulo onde é executada ou de acordo com a parametrização.

☒ **Sintaxe:** OPENFILE(cEmp)

☒ **Parâmetros:**

cEmp	Empresa cujo os arquivos serão re-abertos.
-------------	--

☒ **Retorno:**

Nenhum	.
---------------	---

PERGUNTE()

A função PERGUNTE() inicializa as variáveis de pergunta (mv_par01,...) baseada na pergunta cadastrado no Dicionário de Dados (SX1). Se o parâmetro IAsk não for especificado ou for verdadeiro será exibida a tela para edição da pergunta e se o usuário confirmar as variáveis serão atualizadas e a pergunta no SX1 também será atualizada.

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

☑ **Parâmetros:**

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

☑ **Retorno:**

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

PESQPICT()

A função PESQPICT() retorna a picture definida para um campo especificado no Dicionário de Dados (SX3).

☑ **Sintaxe:** PesqPict(cAlias,cCampo,nTam)

☑ **Parâmetros:**

cAlias	Alias do arquivo
cCampo	Nome do campo
nTam	Opcional, para campos numéricos, será usado como o tamanho do campo para definição da picture. Se não informado, e usado o tamanho padrão no dicionário de dados.

☑ **Retorno:**

String	Picture do campo especificado.
---------------	--------------------------------



Anotações

PESQPICTQT()

A função PESQPICTQT() retorna a picture de um campo numérico referente a uma quantidade, de acordo com o Dicionário de Dados (SX3). Esta função geralmente é utilizada quando há pouco espaço disponível para impressão de valores em relatórios, quando o valor nEdição não é informado, ela tem o comportamento semelhante ao da função "X3Picture", pois busca o tamanho do campo no dicionário de dados.

☑ **Sintaxe:** PesqPictQt(cCampo,nEdição)

☑ **Parâmetros:**

cCampo	Nome do campo a verificar a picture.
nEdição	Espaço disponível para edição.

☑ **Retorno:**

String	Picture ideal para o espaço definido por nEdição, sem a separação dos milhares por vírgula.
---------------	---

POSICIONE()

A função POSICIONE() permite o retorno do conteúdo de um campo de um registro de uma tabela especificado através de uma chave de busca.

☑ **Sintaxe:** Posicione(cAlias, nOrdem, cChave, cCampo)

☑ **Parâmetros:**

cAlias	Alias do arquivo
nOrdem	Ordem utilizada
cChave	Chave pesquisa
cCampo	Campo a ser retornado

☑ **Retorno:**

Indefinido	Conteúdo do campo solicitado.
-------------------	-------------------------------



Importante

A utilização da função POSICIONE() deve ser protegida com GETAREA() / RESTAREA() dependendo da aplicação.

PUTSX1()

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

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

☑ **Parâmetros:**

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

RETINDEX()

A função RETINDEX() restaura os índices padrões de um alias definidos no Dicionário de Dados (SIX).

☑ **Sintaxe:** RETINDEX(cAlias)

☑ **Parâmetros:**

cAlias	Alias de um arquivo do sistema existente no Dicionário de Dados.
---------------	--

☑ **Retorno:**

Numérico	Indica quantos índices padrões o alias especificado possui no Dicionário de Dados.
-----------------	--



Importante

A função RETINDEX() quando utilizada em ambientes TOPCONNECT retorna -1

SIXDESCRICAO()

A função SIXDESCRICAO() retorna a descrição da chave de índice, de acordo com o registro posicionado no SIX e idioma corrente.

☑ **Sintaxe:** SIXDESCRICAO()

☑ **Parâmetros:**

Nenhum	.
---------------	---

☑ **Retorno:**

String	Descrição do índice posicionado no SIX de acordo com o idioma corrente.
---------------	---

Exemplo:

```
User Function <nome-da-função>( cChave, cOrdem )
Local cSixDesc := ""

dbSelectArea("SIX")
dbSetOrder(1)

If dbSeek(cChave+cOrdem)
    cSixDescr := SixDescricao()
EndIf
Return
```

TABELA()

A função TABELA() retorna o conteúdo de uma tabela cadastrada no Arquivo de Tabelas (SX5) de acordo com a chave especificada. Caso a tabela ou a chave especificada não existir será exibido um HELP() padrão do sistema.

☑ **Sintaxe:** Tabela(cTab,cChav,IPrint)

☑ **Parâmetros:**

cTab	Identificação da tabela a pesquisar (deve ser informado como caracter).
cChav	Chave a pesquisar na tabela informada.
IPrint	Indica se deve (.T.) ou não (.F.) exibir o help ou a chave NOTAB se a tabela não existir.

☑ **Retorno:**

String	Conteúdo da tabela na chave especificada. Retorna nulo caso a tabela não exista ou a chave não seja encontrada.
---------------	---

TAMXS3()

A função TAMXS3() retorna o tamanho (total e parte decimal) de um campo especificado no Dicionário de Dados (SX3).

☑ **Sintaxe:** TAMXS3(cCampo)

☑ **Parâmetros:**

cCampo	Nome do campo a ser consultado no Dicionário de Dados (SX3).
---------------	--

☑ **Retorno:**

Array	Array de duas posições contendo o tamanho total e o número de decimais do campo especificado respectivamente.
--------------	---



Anotações

TM()

A função TM() retorna a picture de impressão para valores numéricos dependendo do espaço disponível.

☑ **Sintaxe:** TM(nValor, nEdição, nDec)

☑ **Parâmetros:**

nValor	Valor a ser avaliado.
nEdição	Espaço disponível para edição.
nDec	Número de casas decimais.

☑ **Retorno:**

String	Picture ideal para edição do valor nValor
---------------	---



Importante

Esta rotina leva em consideração duas variáveis:

- MV_MILHAR – Determina se deve haver separação de milhar;
- MV_CENT – Número de casas decimais padrão da moeda corrente.

Para ajustar o valor passado (nValor) ao espaço disponível (nEdição) a função verifica se pode haver separação de milhar, neste caso, a rotina eliminará tantos pontos decimais quantos sejam necessários ao ajuste do tamanho. Caso não seja possível ajustar o valor ao espaço dado, será colocado na picture o caracter de estouro de campo "***". A função também ajusta um valor ao número de decimais (nDec), sempre imprimindo a quantidade de decimais passados no parâmetro.



Anotações

X1DEF01()

A função X1DEF01() retorna o conteúdo da primeira definição da pergunta posicionada no SX1 (caso seja combo) no idioma corrente.

☒ **Sintaxe:** X1DEF01()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

String	Conteúdo da primeira definição da pergunta no idioma corrente.
--------	--

Exemplo:

```
User Function <nome-da-função>( cGrupo, cPerg )
```

```
Local cDef01
```

```
Local cDef02
```

```
Local cDef03
```

```
Local cDef04
```

```
Local cDef05
```

```
dbSelectArea("SX1")
```

```
dbSetOrder(1)
```

```
If dbSeek( cGrupo + cPerg ) // grupo da pergunta + o numero da perg.
```

```
    cDef01 := X1Def01()
```

```
    cDef02 := X1Def02()
```

```
    cDef03 := X1Def03()
```

```
    cDef04 := X1Def04()
```

```
    cDef05 := X1Def05()
```

```
EndIf
```

```
Return
```



Anotações

X1PERGUNT()

A função X1PERGUNT() retorna a descrição da pergunta posicionada no Dicionário de Dados (SX1) para o idioma corrente.

☒ **Sintaxe:** X1PERGUNT()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

String	Descrição da pergunta do Dicionário de Dados (SX1) no idioma corrente.
--------	--

Exemplo:

```
User Function <nome-da-função>( cGrupo, cPerg )
Local cDescr
dbSelectArea("SX1")
dbSetOrder(1)
If dbSeek( cGrupo + cPerg ) // grupo da pergunta + o numero da perg.
    cDescr := X1Pergunt()
EndIf
Return
```

X2NOME()

A função X2NOME() retorna a descrição de uma tabela posicionada no Dicionário de Dados (SX2) no idioma corrente.

☒ **Sintaxe:** X2NOME()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

String	Descrição da tabela posicionada no Dicionário de Dados (SX2) no idioma corrente.
--------	--

Exemplo:

```
User Function <nome-da-função>( )
Local cTabela
dbSelectArea("SX2")
dbSetOrder(1)
If dbSeek( "SA1" )
    cTabela := X2Nome()
EndIf
Return
```

X3CBOX()

A função X3CBOX() retorna o conteúdo de um campo tipo combo posicionado no Dicionário de Dados (SX3) no idioma corrente.

☒ **Sintaxe:** X3CBOX()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

String	Conteúdo do campo do tipo combo posicionado no Dicionário de Dados (SX3) no idioma corrente.
--------	--

Exemplo:

```
User Function <nome-da-função>( )
```

```
Local cTitulo  
Local cDescri  
Local cCombo
```

```
dbSelectArea("SX3")  
dbSetOrder(2)
```

```
If dbSeek( cCampo )  
    cTitulo := X3Titulo()  
    cDescri := X3Descri()  
    cCombo := X3Cbox()  
EndIf
```

```
Return
```

X3DESCRIC()

A função X3DESCRIC() retorna a descrição de um campo posicionado no Dicionário de Dados (SX3) no idioma corrente.

☒ **Sintaxe:** X3DESCRIC()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

String	Descrição do campo posicionado no Dicionário de Dados (SX3) no idioma corrente.
--------	---

Exemplo:

User Function <nome-da-função>()

Local cTitulo
Local cDescri
Local cCombo

dbSelectArea("SX3")
dbSetOrder(2)

If dbSeek(cCampo)
 cTitulo := X3Titulo()
 cDescri := X3Descri()
 cCombo := X3Cbox()
EndIf

Return

X3PICTURE()

A função X3PICTURE() retorna a máscara de um campo contido no Dicionário de Dados (SX3).

☒ **Sintaxe:** X3PICTURE(cCampo)

☒ **Parâmetros:**

cCampo	Nome do campo contido no Dicionário de Dados (SX3).
---------------	---

☒ **Retorno:**

String	Picture do campo informado.
---------------	-----------------------------

Exemplo:

User Function <nome-da-função>(cCampo)

Local cPicture

cPicture := X3Picture(cCampo)
Return cPicture



Anotações

X3TITULO()

A função X3TITULO() retorna o título de um campo posicionado no Dicionário de Dados (SX3) no idioma corrente.

☑ **Sintaxe:** X3TITULO()

☑ **Parâmetros:**

Nenhum	.
--------	---

☑ **Retorno:**

String	Título do campo posicionado no dicionário de dados (SX3) no idioma corrente.
--------	--

Exemplo:

User Function <nome-da-função>()

Local cTitulo

dbSelectArea("SX3")

dbSetOrder(2)

If dbSeek("A1_COD")
 cTitulo := X3Titulo()

EndIf

Return

X3USO()

A função X3USO() verifica se o campo atualmente posicionado no Dicionário de Dados (SX3) está disponível para uso.

☑ **Sintaxe:** X3USO(cUsado, [Modulo])

☑ **Parâmetros:**

cUsado	Conteúdo do campo X3_USADO a ser avaliado.
Modulo	Número do módulo. Caso não seja informado será assumido como padrão o número do módulo corrente.

☑ **Retorno:**

Lógico	Indica se o campo está configurado como usado no Dicionário de Dados (SX3).
--------	---

Exemplo:

```
User Function <nome-da-função>()
```

```
Local lUsado := .F.
```

```
DbSelectArea("SX3")
```

```
DbSetOrder(2)
```

```
DbSeek("A1_COD")
```

```
If X3Uso( SX3->X3_USADO )
```

```
    lUsado := .T.
```

```
EndIf
```

```
Return lUsado
```

X5DESCRI()

A função X5DESCRI() retorna a descrição de um item de uma tabela posicionado no Arquivo de Tabelas (SX5) no idioma corrente.

☒ **Sintaxe:** X5DESCRI()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

String	Descrição do item do Arquivo de Tabelas (SX5) no idioma corrente.
--------	---

Exemplo:

```
User Function <nome-da-função>( cFilial, cTabela, cChave )
```

```
Local cDescr
```

```
dbSelectArea("SX5")
```

```
dbSetOrder(1)
```

```
If dbSeek( cFilial+cTabela+cChave )
```

```
    cDescr := X5Descr()
```

```
EndIf
```

```
Return
```

X6CONTEUD()

A função X6CONTEUD() retorna o conteúdo de um parâmetro posicionado no Dicionário de Dados (SX6) para o idioma corrente.

☑ **Sintaxe:** X6CONTEUD()

☑ **Parâmetros:**

Nenhum	.
--------	---

☑ **Retorno:**

Indefinido	Conteúdo do parâmetro posicionado no Dicionário de Dados (SX6) para o idioma corrente.
-------------------	--



Importante

Utilizar preferencialmente as funções de manipulação de parâmetros GETMV() e suas variantes.

Exemplo:

```
User Function <nome-da-função>( cFilial, cParam )
Local cDescr
Local cConteud

dbSelectArea("SX6")
dbSetOrder(1)

If dbSeek( cFilial+cParm )
    cDescr := X6Descric()
    cDescr += X6Desc1()
    cDescr += X6Desc2()
    cConteud := X6Conteud()
EndIf

Return
```



Anotações

X6DESCRIC()

A função X6DESCRI() retorna o conteúdo da descrição de um parâmetro de acordo com o registro posicionado no Dicionário de Dados (SX6) no idioma corrente.

☑ **Sintaxe: X6DESCRIC()**

☑ **Parâmetros:**

Nenhum	.
--------	---

☑ **Retorno:**

String	Descrição do parâmetro posicionado no Dicionário de Dados (SX6) no idioma corrente.
--------	---



Dica

Para avaliar os conteúdos dos primeiro e segundo complementos da descrição do parâmetro utilize as funções:

- X6DESC01() → retorna o primeiro complemento da descrição.
- X6DESC02() → retorna o segundo complemento da descrição.

As três funções possuem a mesma sintaxe e forma de utilização.

Exemplo:

```
User Function <nome-da-função>( cFilial, cParam )
```

```
Local cDescr  
Local cConteud
```

```
dbSelectArea("SX6")  
dbSetOrder(1)
```

```
If dbSeek( cFilial+cParm )  
    cDescr := X6Descric()  
    cDescr += X6Desc1()  
    cDescr += X6Desc2()  
    cConteud := X6Conteud()
```

```
EndIf  
Return
```


XADESCRI()

A função XADESCRI() retorna o conteúdo da descrição dos folders de acordo com o registro posicionado no Dicionário de Dados (SXA) no idioma corrente.

☒ **Sintaxe:** XADESCRI()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

String	Descrição do folder posicionado no Dicionário de Dados (SXA) no idioma corrente.
--------	--

Exemplo:

```
User Function <nome-da-função>( cFolder, cNumero )
Local cDescr
dbSelectArea("SXA")
dbSetOrder(1)
If dbSeek( cFolder+cNumero ) // alias do folder + numero do folder
    cDescr := XADescr()
EndIf
Return
```

XBDESCRI()

A função XBDESCRI() retorna o conteúdo da descrição de uma consulta de acordo com o registro posicionado no Dicionário de Dados (SXB) no idioma corrente.

☒ **Sintaxe:** XBDESCRI()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

String	Descrição da consulta posicionada no Dicionário de Dados (SXB) no idioma corrente.
--------	--

Exemplo:

```
User Function <nome-da-função>( cAlias )
Local cDescr
dbSelectArea("SXB")
dbSetOrder(1)
If dbSeek( cAlias + "1" )
    cDescr := XBDescr()
EndIf
Return
```

XFILIAL()

A função XFILIAL() retorna a filial utilizada por determinado arquivo.

Esta função é utilizada para permitir que pesquisas e consultas em arquivos trabalhem somente com os dados da filial corrente, dependendo é claro se o arquivo está compartilhado ou não (definição que é feita através do módulo Configurador – Dicionário de Dados (SX2)).

É importante verificar que esta função não tem por objetivo retornar apenas a filial corrente, mas retorná-la caso o arquivo seja exclusivo. Se o arquivo estiver compartilhado, a função xFilial retornará dois espaços em branco.

☒ **Sintaxe: XFILIAL(cAlias)**

☒ **Parâmetros:**

cAlias	Alias do arquivo desejado. Se não for especificado, o arquivo tratado será o da área corrente.
---------------	--

☒ **Retorno:**

Caracter	String contendo a filial do arquivo corrente.
-----------------	---



Anotações

Componentes da interface visual

MSDIALOG()

Define o componente MSDIALOG(), o qual é utilizado como base para os demais componentes da interface visual, pois um componente MSDIALOG() é uma janela da aplicação.

☒ Sintaxe:

```
DEFINE MSDIALOG oObjetoDLG TITLE cTitulo FROM nLinIni,nColIni TO nLiFim,nColFim OF  
oObjetoRef UNIDADE
```

☒ Parâmetros

oObjetoDLG	Posição do objeto Say em função da janela em que ele será definido.
cTitulo	Título da janela de diálogo.
nLinIni, nColIni	Posição inicial em linha / coluna da janela.
nLiFim, nColFim	Posição final em linha / coluna da janela.
oObjetoRef	Objeto dialog no qual a janela será definida.
UNIDADE	Unidade de medida das dimensões: PIXEL

Exemplo:

```
DEFINE MSDIALOG oDlg TITLE cTitulo FROM 000,000 TO 080,300 PIXEL  
ACTIVATE MSDIALOG oDlg CENTERED
```



Anotações

MSGET()

Define o componente visual MSGET, o qual é utilizado para captura de informações digitáveis na tela da interface.

☒ Sintaxe:

@ nLinha, nColuna MSGET VARIABEL SIZE nLargura,nAltura UNIDADE OF oObjetoRef F3 cF3
VALID VALID WHEN WHEN PICTURE cPicture

☒ Parâmetros

nLinha, nColuna	Posição do objeto MsGet em função da janela em que ele será definido.
VARIABEL	Variável da aplicação que será vinculada ao objeto MsGet, que definirá suas características e na qual será armazenado o que for informado no campo.
nLargura,nAltura	Dimensões do objeto MsGet para exibição do texto.
UNIDADE	Unidade de medida das dimensões: PIXEL
oObjetoRef	Objeto dialog no qual o componente será definido.
cF3	String que define a consulta padrão que será vinculada ao campo.
VALID	Função de validação para o campo.
WHEN	Condição para manipulação do campo, a qual pode ser diretamente .T. ou .F., ou uma variável ou uma chamada de função.
cPicture	String contendo a definição da Picture de digitação do campo.

Exemplo:

@ 010,050 MSGET cCGC SIZE 55, 11 OF oDlg PIXEL PICTURE "@R 99.999.999/9999-99";
VALID !Vazio()



Anotações

SAY()

Define o componente visual SAY, o qual é utilizado para exibição de textos em uma tela de interface.

☒ **Sintaxe:**

@ nLinha, nColuna SAY cTexto SIZE nLargura,nAltura UNIDADE OF oObjetoRef

☒ **Parâmetros**

nLinha, nColuna	Posição do objeto Say em função da janela em que ele será definido.
cTexto	Texto que será exibido pelo objeto Say.
nLargura,nAltura	Dimensões do objeto Say para exibição do texto.
UNIDADE	Unidade de medida das dimensões: PIXEL
oObjetoRef	Objeto dialog no qual o componente será definido.

Exemplo:

@ 010,010 SAY cTexto SIZE 55, 07 OF oDlg PIXEL

BUTTON()

Define o componente visual Button, o qual permite a inclusão de botões de operação na tela da interface, os quais serão visualizados somente com um texto simples para sua identificação.

☒ **Sintaxe: BUTTON()**

@ nLinha,nColuna BUTTON cTexto SIZE nLargura,nAltura UNIDADE OF oObjetoRef
AÇÃO AÇÃO

☒ **Parâmetros**

nLinha,nColuna	Posição do objeto Button em função da janela em que ele será definido.
cTexto	String contendo o texto que será exibido no botão.
nLargura,nAltura	Dimensões do objeto Button para exibição do texto.
UNIDADE	Unidade de medida das dimensões: PIXEL
oObjetoRef	Objeto dialog no qual o componente será definido.
AÇÃO	Função ou lista de expressões que define o comportamento do botão quando ele for utilizado.

Exemplo:

010, 120 BUTTON "Confirmar" SIZE 080, 047 PIXEL OF oDlg;
ACTION (nOpca := 1,oDlg:End())

SBUTTON()

Define o componente visual SButton, o qual permite a inclusão de botões de operação na tela da interface, os quais serão visualizados dependendo da interface do sistema ERP utilizada somente com um texto simples para sua identificação, ou com uma imagem (BitMap) pré-definido.

☑ **Sintaxe: SBUTTON()**

DEFINE SBUTTON FROM nLinha, nColuna TYPE N ACTION AÇÃO STATUS OF oObjetoRet

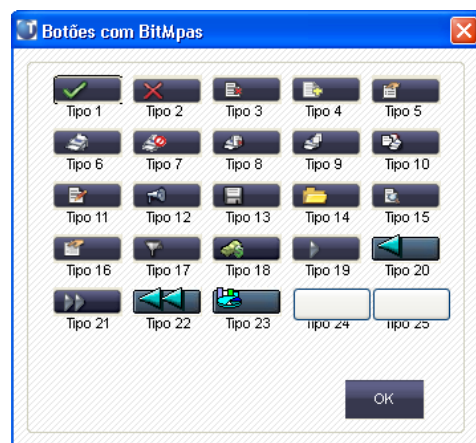
☑ **Parâmetros**

nLinha, nColuna	Posição do objeto sButton em função da janela em que ele será definido.
TYPE N	Número que indica o tipo do botão (imagem) pré-definida que será utilizada.
AÇÃO	Função ou lista de expressões que define o comportamento do botão quando ele for utilizado.
STATUS	Propriedade de uso do botão: ENABLE ou DISABLE
oObjetoRet	Objeto dialog no qual o componente será definido.

Exemplo:

DEFINE SBUTTON FROM 020, 120 TYPE 2 ACTION (nOpca := 2,oDlg:End());
ENABLE OF oDlg

☑ **Visual dos diferentes tipos de botões disponíveis**



CHECKBOX()

Define o componente visual CheckBox, o qual permite a utilização de uma marca para habilitar ou não uma opção escolhida, sendo esta marca acompanhada de um texto explicativo. Difere do RadioMenu pois cada elemento do check é único, mas o Radio permite a utilização de uma lista junto com um controle de seleção.

☒ **Sintaxe:**

```
@ nLinha,nColuna CHECKBOX oCheckBox VAR VARIABEL PROMPT cTexto WHEN WHEN
UNIDADE OF oObjetoRef SIZE nLargura,nAltura MESSAGE cMensagem
```

☒ **Parâmetros:**

nLinha,nColuna	Posição do objeto ComboBox em função da janela em que ele será definido.
oCheckBox	Objeto do tipo CheckBox que será criado.
VARIABEL	Variável do tipo lógico com o status do objeto (.T. – marcado, .F. – desmarcado).
cTexto	Texto que será exibido ao lado do get de marcação.
WHEN	Condição para manipulação do objeto, a qual pode ser diretamente .T. ou .F., ou uma variável ou uma chamada de função.
UNIDADE	Unidade de medida das dimensões: PIXEL
oObjetoRef	Objeto dialog no qual o componente será definido.
nLargura,nAltura	Dimensões do objeto CheckBox.
cMensagem	Texto que será exibido ao clicar no componente.

Exemplo:

```
@ 110,10 CHECKBOX oChk VAR lChk PROMPT "Marca/Desmarca" SIZE 60,007 PIXEL OF oDlg ;
ON CLICK(aEval(aVetor,{|x| x[1]:=lChk}),oLbx:Refresh())
```



Anotações

COMBOBOX()

Define o componente visual ComboBox, o qual permite seleção de um item dentro de uma lista de opções de textos simples no formato de um vetor.

☒ Sintaxe:

@ nLinha,nColuna COMBOBOX VARIABEL ITEMS AITENS SIZE nLargura,nAltura UNIDADE OF oObjetoRef

☒ Parâmetros:

nLinha,nColuna	Posição do objeto ComboBox em função da janela em que ele será definido.
VARIABEL	Variável do tipo caracter que irá receber a descrição do item selecionado no ComboBox.
AITENS	Vetor simples contendo as strings que serão exibidas como opções do ComboBox.
nLargura,nAltura	Dimensões do objeto ComboBox.
UNIDADE	Unidade de medida das dimensões: PIXEL
oObjetoRef	Objeto dialog no qual o componente será definido.

Exemplo:

@ 40, 10 COMBOBOX oCombo VAR cCombo ITEMS aCombo SIZE 180,10 PIXEL OF oFd:aDialogs[2]



Anotações

FOLDER()

Define o componente visual Folder, o qual permite a inclusão de diversos Dialogs dentro de uma mesma interface visual. Um Folder pode ser entendido como um array de Dialogs, aonde cada painel recebe seus componentes e tem seus atributos definidos independentemente dos demais.

☒ Sintaxe:

@ nLinha,nColuna FOLDER oFolder OF oObjetoRef PROMPT &cTexto1,...,&cTextoX UNIDADE SIZE nLargura,nAltura

☒ Parâmetros

nLinha,nColuna	Posição do objeto Folder em função da janela em que ele será definido.
oFolder	Objeto Folder que será criado.
oObjetoRef	Objeto dialog no qual o componente será definido.
&cTexto1,...,&cTextoX	Strings de títulos de cada uma das abas do Folder, sempre precedidas por &. Exemplo: "&Pasta1","&PastaX".
UNIDADE	Unidade de medida das dimensões: PIXEL
nLargura,nAltura	Dimensões do objeto Folder.

Exemplo:

@ 50,06 FOLDER oFld OF oDlg PROMPT "&Buscas", "&Consultas", "Check-&Up / Botões" PIXEL SIZE 222,078



Anotações

RADIO

Define o componente visual Radio, também conhecido como RadioMenu, o qual é seleção de uma opção ou de múltiplas opções através de uma marca para os itens exibidos de uma lista. Difere do componente CheckBox, pois cada elemento de check é sempre único, e o Radio pode conter um ou mais elementos.

☑ Sintaxe:

```
@ nLinha,nColuna RADIO oRadio VAR nRadio 3D SIZE nLargura,nAltura <ITEMS PROMPT>  
cItem1,cItem2,...,cItemX OF oObjetoRef UNIDADE ON CHANGE CHANGE ON CLICK CLICK
```

☑ Parâmetros

nLinha,nColuna	Posição do objeto Radio em função da janela em que ele será definido.
oRadio	Objeto do tipo Radio que será criado.
nRadio	Item do objeto Radio que está selecionado.
3D	Item opcional que define se o RadioButton terá aspecto simples ou 3D.
nLargura,nAltura	Dimensões do objeto Radio.
<ITEMS PROMPT>	Utilizar um dos dois identificadores para definir quais os textos que serão vinculados a cada RadioButton.
cItem1,cItem2,...,cItemX	Texto que será vinculado a cada RadioButton.
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 mudança de um item de um RadioButton para outro.
CLICK	Função ou lista de expressões que será executada na seleção de um item RadioButton.

Exemplo:

```
aAdd( aRadio, "Disco" )  
aAdd( aRadio, "Impressora" )  
aAdd( aRadio, "Scanner" )  
  
@ 30, 10 RADIO oRadio VAR nRadio ITEMS aRadio[1],aRadio[2],aRadio[3] SIZE 65,8 ;  
PIXEL OF ;  
oFld:aDialogs[3] ;  
ON CHANGE ;  
(Iif(nRadio==1,MsgInfo("Opção 1",cAtencao),;  
Iif(nRadio==2,MsgInfo("Opção 2",cAtencao),MsgInfo("Opção 3",cAtencao))))
```

Interfaces de cadastro

AXCADASTRO()

Sintaxe	AxCadastro(cAlias, cTitulo, cVIdExc, cVIdAlt)
Descrição	O AxCadastro() é uma funcionalidade de cadastro simples, com poucas opções de customização.

MBROWSE()

Sintaxe	MBrowse(nLin1, nCol1, nLin2, nCol2, cAlias)
Descrição	A Mbrowse() é uma funcionalidade de cadastro que permite a utilização de recursos mais aprimorados na visualização e manipulação das informações do sistema.

AXPESQUI()

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

- ☒ **Sintaxe: AXPESQUI()**
- ☒ **Parâmetros**

Nenhum	.
---------------	---



Anotações

AXVISUAL()

Função de visualização padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

☑ **Sintaxe:** AXVISUAL(cAlias, nReg, nOpc, aAcho, nColMens, cMensagem, cFunc,; aButtons, IMaximized)

☑ **Parâmetros**

cAlias	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada
nReg	Record number (recno) do registro posicionado no alias ativo.
nOpc	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização).
aAcho	Vetor com nome dos campos que serão exibidos. Os campos de usuário sempre serão exibidos se não existir no parâmetro um elemento com a expressão "NOUSER".
nColMens	Parâmetro não utilizado.
cMensagem	Parâmetro não utilizado.
cFunc	Função que deverá ser utilizada para carregar as variáveis que serão utilizadas pela Enchoice. Neste caso o parâmetro IVirtual é definido internamente pela AxFunction() executada como .T.
aButtons	Botões adicionais para a EnchoiceBar, no formato: aArray[n][1] -> Imagem do botão aArray[n][2] -> bloco de código contendo a ação do botão aArray[n][3] -> título do botão
IMaximized	Indica se a janela deverá ser ou não maximizada

AXINCLUI()

Função de inclusão padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

☑ **Sintaxe:** AxInclui(cAlias, nReg, nOpc, aAcho, cFunc, aCpos, cTudoOk, IF3,; cTransact, aButtons, aParam, aAuto, IVirtual, IMaximized)

☑ **Parâmetros**

cAlias	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada
nReg	Record number (recno) do registro posicionado no alias ativo.
nOpc	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização).
aAcho	Vetor com nome dos campos que serão exibidos. Os campos de usuário sempre serão exibidos se não existir no parâmetro um elemento com a expressão "NOUSER".
cFunc	Função que deverá ser utilizada para carregar as variáveis que serão utilizadas pela Enchoice. Neste caso o parâmetro IVirtual é definido internamente pela AxFunction() executada como .T.
aCpos	Vetor com nome dos campos que poderão ser editados

cTudoOk	Função de validação de confirmação da tela. Não deve ser passada como Bloco de Código, mas pode ser passada como uma lista de expressões, desde que a última ação efetue um retorno lógico: “(Func1(), Func2(), ...,FuncX(), .T.)”
IF3	Indica se a enchoice esta sendo criada em uma consulta F3 para utilizar variáveis de memória
cTransact	Função que será executada dentro da transação da AxFunction()
aButtons	Botões adicionais para a EnchoiceBar, no formato: aArray[n][1] -> Imagem do botão aArray[n][2] -> bloco de código contendo a ação do botão aArray[n][3] -> título do botão
aParam	Funções para execução em pontos pré-definidos da AxFunction(), conforme abaixo: aParam[1] := Bloco de código que será processado antes da exibição da interface. aParam[2] := Bloco de código para processamento na validação da confirmação. aParam[3] := Bloco de código que será executado dentro da transação da AxFunction(). aParam[4] := Bloco de código que será executado fora da transação da AxFunction().
aAuto	Array no formato utilizado pela funcionalidade MsExecAuto(). Caso seja informado este array, não será exibida a tela de interface, e será executada a função EnchAuto(). aAuto[n][1] := Nome do campo aAuto[n][2] := Conteúdo do campo aAuto[n][3] := Validação que será utilizada em substituição as validações do SX3
IVirtual	Indica se a Enchoice() chamada pela AxFunction() utilizará variáveis de memória ou os campos da tabela na edição
IMaximized	Indica se a janela deverá ser ou não maximizada

AXALTERA()

Função de alteração padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

☑ **Sintaxe:** AXALTERA(cAlias, nReg, nOpc, aAcho, aCpos, nColMens, cMensagem,; cTudoOk, cTransact, cFunc, aButtons, aParam, aAuto, IVirtual, IMaximized)

☑ **Parâmetros**

➤ Vide documentação de parâmetros da função AxInclui().

AXDELETA()

Função de exclusão padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

☑ **Sintaxe:** AXDELETA(cAlias, nReg, nOpc, cTransact, aCpos, aButtons, aParam,; aAuto, IMaximized)

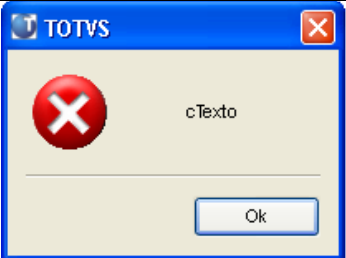
☑ **Parâmetros**

cAlias	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada
nReg	Record number (recno) do registro posicionado no alias ativo.
nOpc	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização).
cTransact	Função que será executada dentro da transação da AxFunction()
aCpos	Vetor com nome dos campos que poderão ser editados
aButtons	Botões adicionais para a EnchoiceBar, no formato: aArray[n][1] -> Imagem do botão aArray[n][2] -> bloco de código contendo a ação do botão aArray[n][3] -> título do botão
aParam	Funções para execução em pontos pré-definidos da AxFunction(), conforme abaixo: aParam[1] := Bloco de código que será processado antes da exibição da interface. aParam[2] := Bloco de código para processamento na validação da confirmação. aParam[3] := Bloco de código que será executado dentro da transação da AxFunction(). aParam[4] := Bloco de código que será executado fora da transação da AxFunction().
aAuto	Array no formato utilizado pela funcionalidade MsExecAuto(). Caso seja informado este array, não será exibida a tela de interface, e será executada a função EnchAuto(). aAuto[n][1] := Nome do campo aAuto[n][2] := Conteúdo do campo aAuto[n][3] := Validação que será utilizada em substituição as validações do SX3
IMaximized	Indica se a janela deverá ser ou não maximizada

Interfaces visuais para aplicações


ALERT()

- ☒ **Sintaxe:** AVISO(cTexto)
- ☒ **Parâmetros**

cTexto	Texto a ser exibido
	


AVISO()

- ☒ **Sintaxe:** AVISO(cTitulo, cTexto, aBotoes, nTamanho)
- ☒ **Retorno:** numérico indicando o botão selecionado.
- ☒ **Parâmetros**

cTitulo	Título da janela
cTexto	Texto do aviso
aBotoes	Array simples (vetor) com os botões de opção
nTamanho	Tamanho (1,2 ou 3)
	

FORMBATCH()


- ☑ **Sintaxe:** FORMBATCH(cTitulo, aTexto, aBotoes, bValid, nAltura, nLargura)
- ☑ **Parâmetros**

cTitulo	Título da janela
aTexto	Array simples (vetor) contendo cada uma das linhas de texto que serão exibidas no corpo da tela.
aBotoes	Array com os botões do tipo SBUTTON(), com a seguinte estrutura: {nTipo,IEnable,{ Ação() }}
bValid	(opcional) Bloco de validação do janela
nAltura	(opcional) Altura em pixels da janela
nLargura	(opcional) Largura em pixels da janela
	

MSGFUNCTIONS()

- ☑ Sintaxe: MSGALERT(cTexto, cTitulo)
- ☑ Sintaxe: MSGINFO(cTexto, cTitulo)
- ☑ Sintaxe: MSGSTOP(cTexto, cTitulo)
- ☑ Sintaxe: MSGYESNO(cTexto, cTitulo)

☑ Parâmetros

cTexto	Texto a ser exibido como mensagem
cTitulo	Título da janela de mensagem
MSGALERT	 A screenshot of a Windows-style dialog box titled 'cTitulo'. It features a yellow warning triangle icon on the left. The text 'cTexto' is displayed in the center. At the bottom right, there is an 'Ok' button.
MSGINFO	 A screenshot of a Windows-style dialog box titled 'cTitulo'. It features a blue information icon (a lowercase 'i' inside a circle) on the left. The text 'cTexto' is displayed in the center. At the bottom right, there is an 'Ok' button.
MSGSTOP	 A screenshot of a Windows-style dialog box titled 'cTitulo'. It features a red 'X' icon on the left. The text 'cTexto' is displayed in the center. At the bottom right, there is an 'Ok' button.
MSGYESNO	 A screenshot of a Windows-style dialog box titled 'cTitulo'. It features a blue question mark icon on the left. The text 'cTexto' is displayed in the center. At the bottom, there are two buttons: 'Sim' (Yes) on the left and 'Não' (No) on the right.

Recursos das interfaces visuais

GDFIELDGET()

A função GDFIELDGET() retorna o conteúdo de um campo especificado em uma grid formada por um objeto do tipo MsNewGetDados() de acordo com a linha da grid desejada.

☑ **Sintaxe:** GDFIELDGET(cCampo, nLinha)

☑ **Parâmetros:**

cCampo	Nome do campo para retorno do conteúdo.
nLinha	Linha da grid que deverá ser avaliada.

☑ **Retorno:**

Indefinido	Conteúdo do campo especificado de acordo com a linha da grid informada.
-------------------	---

GDFIELDPOS()

A função GDFIELDPOS() retorna a posição de um campo especificado em uma grid formada por um objeto do tipo MsNewGetDados().

☑ **Sintaxe:** GDFIELDPOS(cCampo)

☑ **Parâmetros:**

cCampo	Nome do campo a ser avaliado na grid.
---------------	---------------------------------------

☑ **Retorno:**

Numérico	Posição que o campo ocupada na grid. Caso o mesmo não exista será retornado 0.
-----------------	--

GDFIELDPUT()

A função GDFIELDPUT() atualiza o conteúdo de uma grid formada por um objeto do tipo MsNewGetDados() de acordo com o campo e linha da grid especificados.

☑ **Sintaxe:** GDFIELDPUT(cCampo, xConteudo, nLinha)

☑ **Parâmetros:**

cCampo	Nome do campo a ser atualizado.
xConteudo	Conteúdo que será atribuído a célula da grid.
nLinha	Linha da grid que será atualizada.

☑ **Retorno:**

Nenhum	.
---------------	---

GETMARK()

A função GETMARK() é utilizada em conjunto com a função MarkBrow(), para retornar o conjunto de caracteres que serão utilizados para identificar os registros marcados pelo browse.

☑ **Sintaxe:** GETMARK([Upper])

☑ **Parâmetros:**

Upper	Se verdadeiro (.T.) retorna somente caracteres em maiúsculos.
--------------	---

☑ **Retorno:**

String	Conjunto de caracteres que definem a marca que deverá ser utilizada pela MarkBrowse durante o processamento corrente.
---------------	---



Importante

O retorno da função GETMARK() depende do conteúdo atual do parâmetro MV_MARCA.



Importante

É altamente recomendável limpar o conteúdo do campo "marcado" pela MarkBrowse() ao término do processamento, para se evitar problemas com a reutilização da marca após a exaustão das possibilidades de combinação de dois caracteres, o qual é o tamanho padrão do campos utilizados para marcação de registros pela MarkBrowse(), que neste caso somam **1891** possibilidades de "00" a "zz".

Exemplo:

```
Function <nome-da-função>( )
```

```
Local aCampos := { {'CB_OK' , "" } , ;  
{'CB_USERLIB' , 'Usuário' } , ;  
{'CB_TABHORA' , 'Hora' } , ;  
{'CB_DTTAB' , 'Data' } }
```

```
Private cMarca := GetMark()  
Private cCadastro := 'Cadastro de Contrato'  
Private aRotina := { { 'Pesquisar' , 'AxPesqui' , 0, 1 } }
```

```
MarkBrow( 'SCB', 'CB_OK', '!CB_USERLIB', aCampos, , cMarca, 'MarkAll()', , , , 'Mark()' )
```

```
Return
```

MARKBREFRESH()

A função MARKBREFRESH() atualiza a exibição da marca no MarkBrowse(), sendo utilizada quando algum processamento paralelo atualiza o conteúdo do campo definido como controle de marca para os registros em exibição pelo browse.

Este tipo de processamento é comum, e normalmente está associada a clique de "inverter" seleção, ou a opções de "marcar" e "desmarcar" todas.



Importante

A MarkBrowse() atualiza automaticamente a exibição da marca de registros quando utilizado o browse.

☒ **Sintaxe:** MARKBREFRESH()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

Nenhum	.
--------	---

READVAR()

A função READVAR() retorna o nome da variável ou campo associado ao objeto do tipo GET() atualmente selecionado ou em edição.

☒ **Sintaxe:** READVAR()

☒ **Parâmetros:**

Nenhum	.
--------	---

☒ **Retorno:**

String	Nome da variável ou campo associado ao objeto do tipo GET.
--------	--



Anotações
