

Advpl o Manual

Versão 1.0

Índice

A Tecnologia Protheus.....	17
Características da Tecnologia Protheus.....	17
As Camadas do Advanced Protheus.....	17
Servidor de Aplicação.....	18
Terminal Thin-Client.....	18
Dados.....	18
Repositório de Apo's.....	18
Estruturação.....	19
A estrutura de diretórios do Advanced Protheus.....	19
Versão AP5.....	19
Versão AP6.....	19
Aplicativos do Advanced Protheus.....	20
Protheus Server.....	20
Remote.....	20
IDE.....	20
Monitor.....	20
Nomenclaturas Utilizadas.....	21
Topologia de Rede.....	21
Características e Possibilidades de Topologias de Rede.....	21
IDE - Integrated Development Environment.....	23
Características do IDE.....	23
Monitor.....	24
Informações Básicas das Conexões.....	24
Btrieve.....	24
Informações Básicas.....	24
Configuração do Btrieve/PervasiveSQL2000:.....	25
CTree.....	27
Informações Básicas.....	27
Índices.....	27
Sistemas Operacionais e Plataformas (versão AP6 e posteriores).....	28
Sistemas Operacionais e Plataformas.....	28
Plataformas e SO's suportados.....	28
Novos SO's a serem suportados.....	28
AdPI – Guia de Programação.....	29
A Linguagem AdvPI.....	29
Programação Com Interface Própria com o Usuário.....	29
Programação Sem Interface Própria com o Usuário.....	29
Programação por Processos.....	30

Programação de RPC.....	30
Programação Web.....	30
Programação TelNet.....	30
Sintaxe da Linguagem	30
Edição de Programas	30
Criação de um Programa	30
Linhas de Programa	31
Linhas de Comando	31
Linhas de Comentário.....	31
Linhas Mistas	32
Tamanho da Linha.....	32
Estrutura de um Programa	32
A Área de Identificação	33
A Área de Ajustes Iniciais.....	33
O Corpo do Programa.....	34
A Área de Encerramento.....	34
Variáveis de Memória.....	34
Tipos de Dados.....	34
Numérico	34
Lógico	35
Caracter	35
Data	35
Matriz (Array)	35
Bloco de Código	35
Criação e Atribuição de Variáveis.....	36
Matrizes.....	38
Matrizes como Estruturas	39
Cuidados com Matrizes.....	41
Inicializando Matrizes.....	41
Se o tamanho da matriz é conhecido.....	41
Se o tamanho da matriz não é conhecido	42
Blocos de Código.....	42
Um Primeiro Lembrete	42
Outro Lembrete.....	43
Lista de Expressões.....	44
Duas Linhas de Código.....	44
Duas Linha de Código em Uma , Utilizando Ponto-e-Vírgula	44
Convertendo para uma Lista de Expressões	44
Onde Pode-se Utilizar uma Lista de Expressões?.....	45
De Listas de Expressões para Blocos de Código	45
Executando um Bloco de Código	46
Passando Parâmetros	47
Utilizando Blocos de Código	47
Escopo de Variáveis.....	48

O Contexto de Variáveis dentro de um Programa.....	48
Variáveis Locais	49
Variáveis Estáticas	50
Variáveis Privadas.....	50
Variáveis Públicas	51
Diferenciação entre variáveis e nomes de campos	52
Operadores da Linguagem	53
Operadores Comuns.....	53
Operadores Matemáticos	53
Operadores de String.....	54
Operadores Relacionais.....	54
Operadores Lógicos.....	54
Operadores de Atribuição	55
Atribuição Simples	55
Atribuição em Linha	55
Atribuição Composta.....	56
Operadores de Incremento/Decremento.....	56
Operadores Especiais.....	57
Ordem de Precedência dos Operadores	58
Alteração da Precedência.....	58
Macro Substituição	59
Estruturas de Controle.....	60
Controlando o Fluxo	60
Estruturas de Repetição	60
Repetição de Comandos.....	60
O Comando FOR...NEXT	60
O Comando WHILE...ENDDO	62
Estruturas de Decisão.....	63
Desviando a Execução	63
O Comando IF...ENDIF	63
O Comando DO CASE...ENDCASE	64
Informações Adicionais.....	65
Lista de Palavras Reservadas.....	65
Técnicas de Programação Eficiente em AdvPI.....	66
Criação de Funções Segundo a Necessidade	67
Codificação Auto-Documentável.....	68
Utilização de Soluções Simples.....	68
Opção por Flexibilidade	69
Opção da Praticidade ao Drama.....	69
Utilização de Operadores de Incremento/Decremento.....	69
Evitar Passos Desnecessários	70
Utilização de Alternativas.....	71
Utilização de Arquivos de Cabeçalho Quando Necessário	72
Constantes em Maiúsculo	72

Utilização de Identação.....	72
Utilização de Espaços em Branco.....	73
Quebra de Linhas Muito Longas.....	73
Capitulação de Palavras-Chave	74
Utilização da Notação Húngara	74
Utilização de Nomes Significantes para Variáveis	74
Utilização de Comentários	74
Criação de Mensagens Sistêmicas Significantes e Consistentes	75
Evitar Abreviação de Comandos em 4 letras.....	75
Evitar "Disfarces" no Código	75
Evitar Código de Segurança Desnecessário.....	76
Isolamento de Strings de Texto	77
Tabela de Pictures de Formatação	77
Comando SAY/PSAY	77
Comando GET.....	77
Programação do AdvPI para o ERP Siga Advanced Protheus	78
Prefácio	78
O Que é Fazer um Programa com Inteligência.....	79
Programando Simples, mas Certo.....	80
Erros que Podem ser Evitados.....	80
A Importância de Programas Documentados.....	81
Cabeçalho de Programa / Função.....	81
Criação de Variáveis.....	82
A Declaração.....	82
O Tipo de Variável.....	82
Tipos Existentes.....	82
Inicialização.....	83
Padronização de Variáveis.....	83
Criando uma Variável Utilizando a Função CRIAVAR()	83
Variáveis de Relatórios	84
Cuidados com o Posicionamento de Registros.....	85
Função Posicione	86
Função Existcpo	86
Restaurando Índice e limpando filtros	86
Outros Cuidados.....	87
Confrontando relatórios e consultas.....	87
Problemas com Looping de Programas	87
Manipulação de Arquivos Externos ao Protheus.....	88
Desenvolvendo Telas.....	89
Salvando Array's padrões	90
Pontos de Entrada	90
Objetivo dos Pontos de Entrada.....	90
Quando criar um Ponto de Entrada.....	90
Sintaxe para criar um Ponto de Entrada:.....	91

Procedimentos para sua criação	91
Contabilizando	91
A Função HeadProva.....	92
A função DetProva()	92
A função RodaProva()	92
Atualizando SX (ATUSX).....	93
Exemplo de atualização no SX:	93
SX1 - Parâmetros Genéricos	93
SX2 - Mapeamento dos arquivos	93
SX3 - Dicionário de Dados.....	94
SX5 - Tabelas Genéricas	94
SX6 - Parâmetros.....	94
SX7 – Gatilhos	94
SXE/SXF - Sequência. de documentos.....	94
SINDEX- Arquivo de Índices.....	94
Controle de Transação (TTS)	94
O que é	94
Quando usar	94
Como usar	94
Onde não usar	95
Comandos Definidos pelo Usuário (UDC´s).....	95
Uso de Strings	96
Conceito de Filial e Compartilhamento de Arquivos	97
Arquivos Compartilhados	97
Arquivos Exclusivos	97
Técnicas para Filtragem	98
Chaves Primárias	98
Chaves Estrangeiras.....	98
Integridade Referencial.....	98
Utilizando Rotinas Automáticas	99
Controle de Semáforo.....	99
Atualização do SourceSafe.....	100
Procedimentos de Localizações	100
Procedimentos a serem cumpridos em alteração / desenvolvimento de programas :	100
Programando com Schedule de Relatórios	101
Modelos de Programas Padronizados.....	101
Modelo 1	102
Modelo 2	104
Modelo 3	112
Modelos de Relatórios.....	126
Modelos de Cálculos.....	130
Funções básicas da Linguagem	139
AAdd	139

ACopy	140
AClone	141
ADel	142
ADir	142
AEval	143
AFill	145
AIns	146
Alias	146
Append From	147
Array	149
AScan	150
ASize	152
ASort	153
ATail	154
BTVCanOpen	154
BTVCreateDDFs	155
BTVDropIdxs	156
BTVTables	157
CDow	158
CMonth	158
Commit	159
Copy To	160
Copy File	161
Copy Structure	162
CPYS2T	163
CPYT2S	164
CTreeDelIdxs	164
CTreeDelInt	165
CurDir	166
Date	167
Day	167
DBAppend	168
DBCclearAllFilter	169
DBCclearFilter	170
DBCcloseAll	170
DBCcloseArea	171
DBCclearIndex	171
DBCommit	172
DBCommitAll	172
DBCcreate	173
DBCcreateIndex	174
DBDelete	175
DBEval	175
DBF	177

DBFieldInfo.....	177
DBFilter.....	178
DBGoBottom	179
DBGoTo.....	180
DBGoTop.....	180
DBInfo	181
DBOrderInfo.....	182
DBOrderNickName.....	183
DBRecall.....	184
DBRecordInfo	184
DBReindex	185
DBRLock.....	186
DBRLockList	187
DBRunLock	187
DbSeek	188
DBSetDriver.....	190
DBSetFilter.....	190
DBSetIndex	191
DBSetNickName	192
DBSetOrder	193
DBSkip	194
DBStruct.....	195
DBUnlock	195
DBUnlockAll.....	196
Delete	196
Deleted.....	198
DevOutPict.....	198
DevPos	199
Directory	200
DirRemove.....	201
DiskSpace	202
Dow.....	202
Dtoc.....	203
Dtos.....	204
Eject.....	205
ElapTime	205
FClose	206
FCreate.....	207
FErase.....	208
FError.....	209
FieldBlock.....	210
FieldWbl	211
File.....	212
FOpen	213

FOpenPort.....	214
FRead.....	216
FReadStr	217
Frename	218
FSeek	219
FWrite	220
GetImpWindows.....	222
GetPortActive	222
GetClientDir	223
Header	223
IndexKey.....	224
IndexOrd.....	225
InitPrint	225
IsPrinter	226
IsCisaSyncOn	227
Locate	227
LUpdate.....	229
MakeDir	229
MemoLine	230
MemoRead.....	231
MemoWrite	232
MLCount.....	232
Month	233
MsCompress.....	234
MsCRC32.....	235
MsCRC32Str.....	236
MsDecomp.....	236
OrdCondSet.....	237
OrdCreate	238
OrdDescend.....	240
OrdKey	241
OrdListAdd.....	242
PRow.....	243
Pack	244
PCol.....	244
PreparePrint	245
PrnFlush	246
RDDSetDefault.....	247
RealRDD.....	247
Recall	248
RecSize	249
ReIndex.....	250
Replace.....	250
RLock.....	252

Seconds.....	253
Seek	253
Select.....	254
Set Filter.....	255
Set Index	256
Set Order	257
SetPrc	258
Skip	258
SplitPath.....	260
TCConType.....	261
TCDelFile.....	262
TCCGenQry	262
TCIsvLock	263
TCRefresh.....	263
TCSetBuff.....	264
TCSetConn.....	264
TCSetDummy.....	265
TCSetField.....	266
TCSpExec.....	267
TCSpExist.....	268
TCSqlError	269
TCSrvType	269
TCSysExe	270
;TCUnLink.....	271
TCVUnLock.....	271
TCVLock	272
Time.....	273
UnLock.....	274
UpdateIntName	274
Use	275
Used.....	277
Year.....	277
ZAP.....	278
Classes de Interface Visual.....	279
tSrvObject.....	279
Características	279
Propiedades.....	279
Métodos	280
SetFocus	280
Hide	280
Show.....	280
Enable	280
Disable	281
Refresh.....	281

Classes de Janelas.....	281
tWindow.....	281
Hierarquia.....	281
Características	281
Propriedades.....	282
Métodos	282
New.....	282
Activate.....	283
End.....	284
Center.....	284
Exemplo.....	284
TDialog.....	284
Hierarquia.....	285
Características	285
Propriedades.....	285
Métodos	285
New.....	285
Activate.....	286
End.....	287
MSDialog.....	287
Hierarquia.....	287
Características	287
Propriedades.....	287
Métodos	287
New.....	287
Classes Auxiliares.....	289
tFont	289
Hierarquia.....	289
Descrição	289
Métodos	289
New.....	289
Exemplo	290
Classes de Componentes	290
tControl	290
Hierarquia.....	290
Características	290
Propriedades.....	290
Métodos	291
SetFocus.....	291
Classes de Componentes Visuais	291
tButton	291
Hierarquia.....	291
Descrição	291
Propriedades.....	291

Metodos	292
New.....	292
Exemplo	293
tCheckBox.....	293
Hierarquia.....	293
Descrição	293
Métodos	293
New.....	293
Exemplo	294
tComboBox.....	295
Hierarquia.....	295
Descrição	295
Propriedades	295
Métodos	295
New.....	295
Select.....	296
Exemplo	297
tGet	297
Hierarquia.....	297
Descrição	297
Propriedades	297
Métodos	298
New.....	298
Exemplo	299
tGroup	299
Hierarquia.....	299
Descrição	300
Métodos	300
New.....	300
Exemplo	300
tListbox.....	301
Hierarquia.....	301
Descrição	301
Parâmetros.....	301
Métodos	301
New.....	301
Select.....	302
Add.....	303
Modify	303
Del.....	303
Len.....	304
Reset	304
Exemplo	304
tMeter	305

Hierarquia.....	305
Descrição	305
Parâmetros.....	305
Métodos	305
New.....	305
Set.....	306
Exemplo	306
tMultiget.....	307
Hierarquia.....	307
Descrição	307
Propriedades.....	307
Métodos	308
New.....	308
EnableVScroll.....	309
EnableHScroll.....	309
Exemplo	310
tPanel.....	310
Hierarquia.....	310
Descrição	310
Métodos	310
New.....	310
Exemplo	311
tRadMenu	311
Hierarquia.....	311
Descrição	311
Propriedades.....	312
Métodos	312
New.....	312
EnableItem.....	313
Exemplo	313
tSay.....	313
Hierarquia.....	313
Descrição	314
Parâmetros.....	314
Métodos	314
New.....	314
SetText.....	315
Exemplo	315
tScrollbar.....	316
Hierarquia.....	316
Descrição	316
Métodos	316
New.....	316
Exemplo	316

Infra-estrutura	318
MsGetDados	318
MsGetDb	322
MsmGet	326
MBrowse	329
Funções.....	330
AllGroups.....	330
AllUsers	331
APMsgAlert	333
APMsgInfo	333
APMsgNoYes	334
APMsgStop.....	334
APMsgYesNo	335
APMsgYesNo	335
Cabec.....	336
Capital	338
CloseBrowse.....	338
Conpad1	339
Enchoicebar.....	339
FileNoExt.....	340
Final	341
FTPConnect	341
FTPDirChange.....	342
FTPDiretory	343
FTPDisconnect	343
FTPDownload.....	344
FTPErase.....	344
FTPGetCurDir	345
FTPRenameFile.....	345
FTPUpload	346
FunDesc	347
FunName	347
GetCountryList.....	348
GetMark.....	348
GetMv	349
IncProc	350
IncRegua.....	350
IndRegua	352
MarkBRefresh.....	353
MarkBrow.....	353
Ms_Flush.....	354
MsAppend.....	356
MsCopyFile	356
MsCopyTo.....	357

MsCreate.....	358
MsErase.....	358
MsFile.....	359
MsRename.....	360
MsUnlock.....	360
OurSpool.....	361
Pergunte	362
Processa.....	363
ProcRegua	364
PswAdmin.....	364
PswID	365
PswName	365
PswOrder	366
PswRet.....	367
PswSeek	367
ReadVar.....	368
RetAcsName.....	368
RetExtHlp	369
RetExtHls	369
RetExtHpr	370
RetExtMnu	370
RetFileName.....	371
Roda.....	371
RptStatus	373
SetDefault.....	374
SetPrint.....	376
SetRegua	378
SixDescricao.....	380
VerSenha	380
X1Def01	381
X1Def02	381
X1Def03	382
X1Def04	382
X1Def05	383
X1Pergunt.....	383
X2Nome.....	384
X3CBox.....	385
X3Descric	385
X3Picture.....	386
X3Titulo.....	386
X3Uso	386
X5Descri.....	387
X6Conteud.....	388
X6Desc1	388

<i>AdvPl</i>	16
X6Desc2	389
X6Descric	389
XADescric	390
XBDescr	390

A Tecnologia Protheus

O Advanced Protheus é uma nova tecnologia desenvolvida sobre o sistema Advanced, que teve toda a inteligência dividida em duas camadas: Servidor de Aplicação (Protheus Server) e Interface (Remote). Ou seja, uma aplicação 32 bits que se encarrega do gerenciamento das conexões, da execução do código AdvPI e do acesso aos recursos de banco de dados (ADS, Btrieve, CTree ou TopConnect), e uma aplicação thin-client que efetua apenas a interface com o usuário.

Características da Tecnologia Protheus

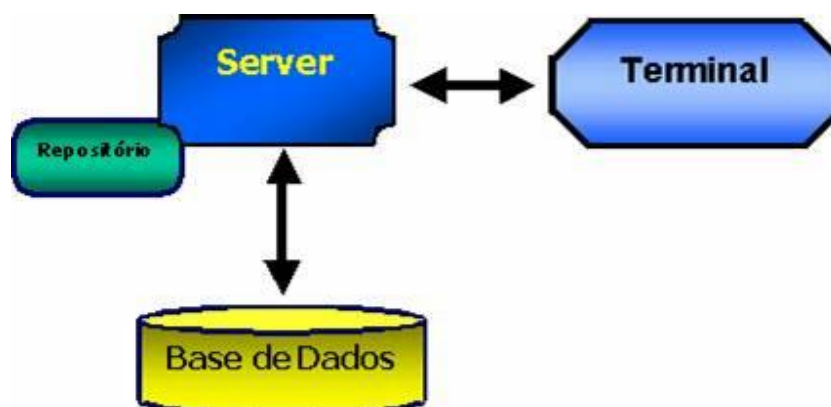
As principais características da tecnologia Protheus são:

- Possibilidade de grande variação de topologias de rede e processamento distribuído
- Baixo tráfego de rede entre o AP5 Server e o AP5 Remote
- Utilização de configurações, possibilitando o uso de conexões simultâneas através de protocolos diferentes e o acesso a diferentes repositórios de APO's e diretórios (o que permite o uso de diferentes idiomas, versões, etc, acessando a mesma base de dados)
- Diferentes possibilidades de impressão de relatórios. No Advanced Protheus pode-se imprimir os relatórios de três formas:
 1. Em disco
 2. Via Windows
 3. Direto na Porta

As impressões efetuadas via Windows ou diretamente na porta podem ser impressas no servidor (o que evitará o tráfego de rede do relatório impresso) ou na estação. As impressões efetuadas em disco também evitam o tráfego de rede. Os relatórios em disco são sempre mantidos no servidor e somente as páginas requisitadas são enviadas à estação.

- Os arquivos de banco de dados são sempre abertos no servidor. Entretanto, arquivos texto podem ser abertos na estação com a função FOpen. A referência a paths que não contiverem uma letra de drive (por exemplo, "\DADOS\ARQ.TXT"), são consideradas a partir do servidor no diretório informado na seção "RootPath" do arquivo de configurações do Protheus Server
- Não existe o conceito de "módulos" de sistema. Existe o conceito de programa inicial, de onde a execução do Remote será iniciada (e os outros APO's utilizados serão carregados e descarregados dinamicamente de acordo com sua chamada a partir deste). Isso permite que rotinas que eram de "módulos" diferentes sejam executadas diretamente de um mesmo menu de usuário

As Camadas do Advanced Protheus



O Advanced Protheus é dividido em quatro camadas para a operação são elas:

- Servidor de Aplicação
- Terminal Thin-Client
- Dados
- Repositório de APO's

Servidor de Aplicação

O Protheus Server é a aplicação encarregada da compilação e da execução do código em AdvPI, no qual o sistema Siga Advanced está escrito a partir da versão 5.07. Na linguagem AdvPI, as rotinas são mantidas em APO's (Advanced Protheus Objects) individuais em repositórios. Isso permite que as rotinas sejam carregadas/descarregadas dinamicamente da memória da máquina onde o Protheus Server está sendo executado, ou seja, de acordo com a necessidade de execução dos Terminais conectados, e facilita a atualização após correções de não-conformidades ou criação de melhorias, pois apenas os APO's modificados necessitam ser atualizados. Desse modo, a performance é alta e não requer muitos recursos da máquina para a execução do servidor.

Terminal Thin-Client

O Remote é a aplicação encarregada da interface com o usuário. Não existe processamento local, por isso o tráfego de rede entre o Terminal e o Servidor de Aplicação é baixo, tratando-se apenas de comandos para o desenho das telas e do tratamento do teclado e mouse.

Dados

O acesso aos dados é efetuado pelo Servidor de Aplicação utilizando as seguintes bases de dados: ADS, Btrieve, CTree e TopConnect (para padrão SQL). Para bases de dados SQL, existe total suporte a Stored Procedures. No Protheus, todas as bases de dados têm suporte a controle de transação.

Repositório de Apo's

É no repositório que se encontram os programas escritos em AdvPI que serão carregados para a execução de determinada tarefa. É através do repositório de Apo's que pode-se incluir novas customizações no sistema.

Estruturação

A estrutura de diretórios do Advanced Protheus

A estrutura de diretórios do Advanced Protheus depende da versão instalada.

Versão AP5

\ap5\	Diretório inicial do Protheus. É a partir deste diretório que o sistema irá localizar os caminhos informados em parâmetros, customizações, etc.
\ap5\apo\	Diretório onde serão localizados os arquivos de repositório de APO's.
\ap5\bin\	Diretório onde são localizados os arquivos do núcleo do Protheus: executáveis, bibliotecas de carga dinâmica (DLL's) e arquivos de configuração do sistema.
\ap5\sigadv\	Similar ao \SIGAADV\ das versões Advanced 2.0x/4.0x. É o diretório onde se encontram os arquivos de configuração do sistema ERP Advanced, arquivos de menus, etc. É também o diretório inicial de execução no Remote.
\ap5\dadosadv\	Similar ao \DADOSADV\ das versões Advanced 2.0x/4.0x. É o diretório onde se localizam os arquivos de base de dados para versões não SQL.
\ap5\relato\	Similar ao \RELATO\ das versões Advanced 2.0x/4.0x. Diretório para gravação de arquivos de impressão em disco.
\ap5\cprova\	Similar ao \CPROVA\ das versões Advanced 2.0x/4.0x. Diretório para gravação de arquivos de contabilização.
\ap5\ixbpad\	Diretório de localização de programas de exemplo escritos em AdvPI, nos padrões definidos pela Microsiga Software S.A.
\ap5\include\	Diretório de arquivos de inclusão padrão (extensão .CH) necessários para a compilação de programas escritos em AdvPI.
\ap5\util\	Diretório de ferramentas adicionais do Protheus.

Versão AP6

\ap6\	Diretório inicial do Protheus. É a partir deste diretório que o sistema irá localizar os caminhos informados em parâmetros, customizações, etc.
\ap6\apo\	Diretório onde serão localizados os arquivos de repositório de APO's.
\ap6\bin\server\	Diretório onde são localizados os arquivos do núcleo do Protheus Server: executáveis, bibliotecas de carga dinâmica (DLL's) e arquivos de configuração.
\ap6\bin\remote\	Diretório onde são localizados os arquivos das aplicações clientes (Remote, IDE, Monitor, etc): executáveis, bibliotecas de carga dinâmica (DLL's) e arquivos de

	configuração.
\ap6\sigadv\	Similar ao \SIGAADV\ das versões Advanced 2.0x/4.0x. É o diretório onde se encontram os arquivos de configuração do sistema ERP Advanced, arquivos de menus, etc. É também o diretório inicial de execução no Remote.
\ap6\dadosadv\	Similar ao \DADOSADV\ das versões Advanced 2.0x/4.0x. É o diretório onde se localizam os arquivos de base de dados para versões não SQL.
\ap6\relato\	Similar ao \RELATO\ das versões Advanced 2.0x/4.0x. Diretório para gravação de arquivos de impressão em disco.
\ap6\cprova\	Similar ao \CPROVA\ das versões Advanced 2.0x/4.0x. Diretório para gravação de arquivos de contabilização.
\ap6\ixbpad\	Diretório de localização de programas de exemplo escritos em AdvPI, nos padrões definidos pela Microsiga Software S.A.
\ap6\include\	Diretório de arquivos de inclusão padrão (extensão .CH) necessários para a compilação de programas escritos em AdvPI.

•

Estas são as estruturas para uma instalação padrão do Protheus de acordo com a versão utilizada. Porém a localização de instalação pode variar de acordo com o local de instalação.

Aplicativos do Advanced Protheus

O Advanced Protheus possui, basicamente, quatro aplicativos utilizados com diferentes finalidades. São eles:

Protheus Server

Trata-se do servidor de aplicação do Advanced Protheus esta é a parte do sistema que será executada no Servidor e será responsável pela comunicação entre o Cliente, a Base de Dados e o Repositório de Apo's. O nome do executável depende da versão e sistema operacional utilizados, por exemplo: AP5SRV.EXE ou AP6SRVWIN.EXE.

Remote

É o Remote que utilizamos para interagir com todo o sistema, ele poderá ser instalado de duas maneiras, no servidor ou na própria estação de trabalho. O nome do executável depende da versão utilizada, por exemplo: AP5RMT.EXE ou AP6RMT.EXE.

IDE

Trata-se do ambiente de desenvolvimento integrado do Advanced Protheus. É através deste aplicativo que todos os acessos aos repositórios de Apo's (compilação de customizações, visualização de funções existentes etc.) são efetuados, e também é a ferramenta para desenvolvimento e depuração de aplicações/customizações. O nome do executável depende da versão utilizada, por exemplo: AP5IDE.EXE ou AP6IDE.EXE.

Monitor

Esta ferramenta permite a interação com os usuários conectados ao sistema: Analisar os programas em uso, derrubar conexões pendentes, enviar mensagens ao usuários etc. O nome do executável depende da versão utilizada, por exemplo: AP5MONIT.EXE ou AP6MONIT.EXE

Alem destas aplicações, o Siga Advanced Protheus conta ainda com outros utilitários diversos, como o Diff (utilizado para comparação de arquivos texto) e Dump (utilizado para edição de arquivos binários).

Nomenclaturas Utilizadas

Estas são as principais nomeclaturas utilizadas no Advanced Protheus:

Build: Versão completa do sistema com seus executáveis, DLL's e RPO completo. O build do sistema pode ser identificado através da opção *Mi scel âneas / Sobre* dentro dos módulos do sistema.

RPO: É o arquivo binário de repositório de APO's, com o código AdvPI.

Patch: Arquivo binário semelhante ao repositório contendo apenas atualizações de APO's, correções disponibilizadas pela Microsiga Software S.A., que será aplicado no repositório através do IDE.

Versão Master: Mesma definição de build porém neste caso a versão será utilizada como referência para a geração de atualizações do sistema (patch's). Quando gerada, a versão é encaminhada a todos os envolvidos em processos de implantação/utilização do sistema via correio ou disponibilizada no site de FTP do Protheus.

•

A definição dos nomes dos arquivos dos repositórios de APO's e Patch's seguem o mesmo padrão (diferenciando-se apenas na extensão Patch=.PAT e repositório=RPO) e é efetuada da seguinte maneira AP12345.RPO:

1 – (D)bf, (T)op, (A)ds, (B)trieve, (C)Tree;

2 – (P)ortuguese, (E)nglish, (S)panish;

3 – Versão;

4 – Versão;

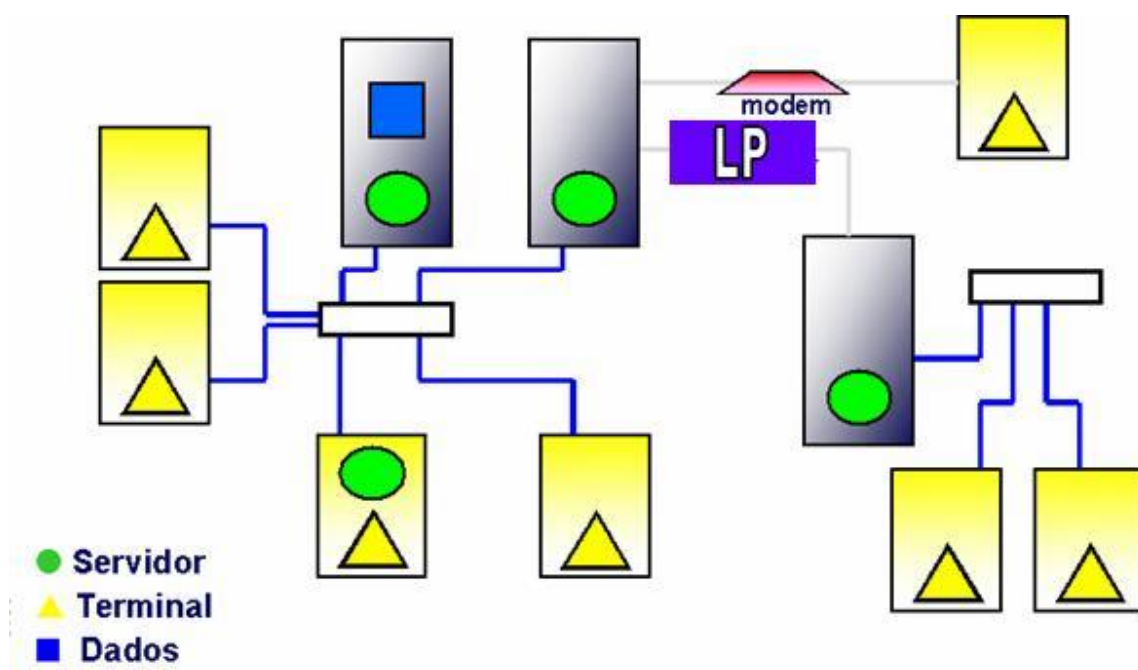
5 – Versão;

Por exemplo, APBP609.RPO será um repositório de APO's para base de dados BTrieve, idioma Português e versão 609.

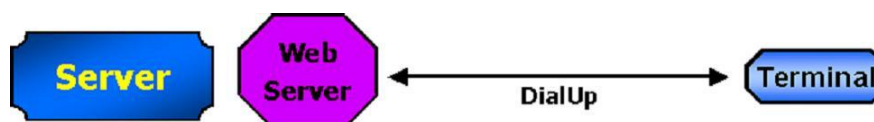
Topologia de Rede

Devido à sua divisão em camadas, a tecnologia Protheus permite montar o ambiente de execução com diferentes topologias, priorizando a execução, o tráfego de rede ou os recursos individuais das máquinas existentes, o que torna o processamento distribuído. Essa é uma das grandes vantagens da tecnologia Protheus.

Características e Possibilidades de Topologias de Rede



- Pode-se manter um ou mais servidores de aplicação do Protheus (Protheus Server).
- Um Protheus Server pode estar na mesma máquina que o gerenciador de banco de dados.
- Máquinas melhores ou com mais recursos podem ter um Protheus Server sendo executado localmente (na estação) visando priorizar o processamento local. Porém há um aumento do tráfego de rede entre o gerenciador de banco de dados e o Protheus Server local.
- Do mesmo modo, pode-se executar servidores remotamente priorizando o processamento de uma ou mais estações.
- Utilização do protocolo TCP-IP para as conexões. Na versão AP5 pode-se utilizar também o protocolo NAMED PIPES.
- Pode-se disponibilizar conexões via Internet através da montagem de um Web Server na empresa ou através de provedores de acesso (como a UOL e o ZAZ). Neste caso, o Protheus Server pode ser executado no provedor ou na empresa utilizando uma LP para a conexão ao provedor de acesso.
- A Internet é utilizada apenas como o meio de comunicação, ou seja, a conexão TCP-IP entre o Remote e o Protheus Server. Além da segurança oferecida pelo sistema Advanced, a segurança deverá ser oferecida pela conexão. Por isso, é aconselhável que a máquina onde o servidor Web esteja sendo executado não seja o mesmo do servidor da banco de dados.



- O Protheus Server também pode ser executado como um servidor Internet, HTTP e/ou FTP.
- Pode-se manter uma página para o download do Remote, de modo que os usuário remotos possam efetuar o download para conectar-se e utilizar o sistema. Na versão AP6 pode-se configurar o Remote para se atualizar automaticamente a partir do Protheus Server sendo executado como um servidor FTP.

-

A definição da melhor topologia para execução é um passo importante da implementação da tecnologia Protheus, pois influenciará totalmente na performance. O maior tráfego de rede está localizado entre o Protheus Server e o Banco de Dados, já que o tráfego entre o Protheus Server e o Remote limita-se às informações para montagem de telas e controle de teclado e mouse. Desta forma, dependendo do tipo e da velocidade da conexão, pode se tornar inviável manter um Protheus Server em uma cidade conectado ao banco de dados em outra. Por outro lado, caso existam muitas conexões nesta outra cidade, ou caso estas conexões executem processamentos pesados, priorizar o processamento disponibilizando um Protheus Server para ser executado localmente nesta cidade pode ser uma boa solução.

IDE - Integrated Development Environment

O IDE (Integrated Development Environment) é uma ferramenta de edição, compilação e depuração de erros. É através do IDE que o sistema ERP Siga Advanced é desenvolvido. Também é através do IDE que os analistas da Microsiga e os usuários do Protheus podem criar e manter suas rotinas específicas. O IDE é o único modo de compilar os arquivos de programas escritos em AdvPI para a geração dos APO's no repositório.

Como ferramenta de edição e depuração, engloba todos os recursos disponíveis nas melhores ferramentas de desenvolvimento do mercado.

Características do IDE

- Para a execução do IDE não é necessário conectar-se ao Protheus Server, exceto nas operações de atualização ou consulta de um repositório (compilação, obtenção do Mapa de Objetos, aplicação de patch's, etc) e durante o processo de depuração.
- Não é aconselhável executar o IDE remotamente (via Internet ou via modem), pois a conexão do Remote ao Protheus, quando efetuada através do IDE, é mais lenta.
- Os passos para o desenvolvimento de programas em AdvPI utilizando o IDE são:
 - 1.Criação do código através do editor. Na linguagem AdvPI, os analistas e os usuários do Protheus têm todos os recursos disponíveis para o desenvolvimento de suas rotinas. E diferentemente do antigo RDMAKE, não é mais uma linguagem interpretada. Por isso, a performance das rotinas específicas é dez vezes maior do que era nos RDMAKES antigos.
 - 2.Montagem do Grupo de Projetos. O Grupo de Projetos é um gerenciador existente dentro do IDE, onde o usuário pode manter os arquivos de código separados por projetos e pastas. Com um grupo de projetos os arquivos podem ser organizados de uma forma lógica, em projetos e pastas.
 - 3.Compilação. Durante a compilação, os arquivos são enviados ao Protheus Server. Toda a compilação e a gravação no repositório são efetuadas no servidor.
 - 4.Depuração. O IDE permite aos usuários depurar as rotinas criadas, executando-as linha a linha ou em modo de animação. Permite visualizar informações como variáveis em diferentes escopos, pilha de chamadas, lista de break points, etc. Exatamente como as melhores ferramentas de desenvolvimento existentes no mercado. No IDE pode-se depurar pontos de entrada simplesmente colocando-se um ponto de parada (break point) em uma linha qualquer do código do ponto de entrada;

Monitor

O Monitor é utilizado para monitorar as conexões de terminais ao Protheus Server. Através dele pode-se:

- Verificar as conexões ativas.
- Enviar mensagens para uma ou mais conexões.
- Desabilitar novas conexões. Isto é útil quando se precisa efetuar alguma manutenção e se precisa evitar que outros usuários se conectem.

Informações Básicas das Conexões

- Usuário. É o nome do usuário na rede local. Para conexões remotas este nome está em branco.
- Computador. Nome da máquina onde o Remote está sendo executado.
- Conexão. Indica a data e hora de início da conexão do Remote.
- Tempo de Uso. Indica o tempo em horas, minutos e segundos desde que o Remote se conectou.
- Programa Inicial. É o nome do programa inicial (APO) com o qual o Remote iniciou a execução.
- Environment. Nome do ambiente sendo utilizado pelo terminal.

Btrieve

Informações Básicas

- Para manipulação de tabelas Btrieve o driver utilizado é "BTVC DX";
- Para programar algo específico para o Btrieve pode-se utilizar o teste "#ifdef BTV";
- A extensão padrão das tabelas é ".dat";
- Os índices são criados no mesmo arquivo de dados (".dat");
- As manipulações e visualizações de dados devem ser feitas através do "APSdu", pois as antigas ferramentas são incompatíveis;
- O Btrieve versão 6.15 não precisa ser instalado, pois as DLLs necessárias são disponibilizadas junto com o Protheus. Os arquivos necessários ficam no diretório "BIN" (wbtrv32.dll, w32mkrc.dll, wbtrvres.dll e w32mkde.exe). O funcionamento é praticamente igual ao Btrieve Server, portanto pode-se executar os testes na versão 6.15 normalmente;
- O w32mkde continua um período em execução após o término do Protheus, pois se o Protheus for executado novamente não é necessário seu reinício. Quando o usuário desejar renomear o diretório

"BIN", o mesmo não será permitido por esse motivo, deve-se portanto finalizar a execução do mesmo;

- As informações das tabelas, campos e índices são armazenados no diretório "DDF", criado abaixo do "RootPath", através dos arquivos (Field.btv, File.btv, Finfo.btv, Iinfo.btv e Index.btv). Se estes arquivos forem apagados as tabelas serão recriadas e todos os dados serão PERDIDOS. Não se pode copiar uma tabela com estrutura diferente para este diretório, pois seus dados devem ser atualizados nos arquivos do DDF também. Como os dados e o diretório DDF devem estar "sincronizados" os arquivos do DDF devem ser incluídos no esquema de "backup" dos dados;
- As tabelas só podem ter "um" campo do tipo memo e este campo deve ser o último, por isso na hora da criação da tabela o Protheus automaticamente desloca o campo memo para o final e mostra uma mensagem de aviso;
- Para apagar os índices, entrar em APSdu, abrir a tabela e escolher Index/erase all. Ele apagará todos os índices da tabela e também a sua definição no DDF. Para fazer via programa, selecione a tabela e chame a função <@>BTVDropIdxs(). Portanto aconselha-se utilizar o índice do tipo permanente somente se o mesmo for utilizado posteriormente (outras aberturas da tabela) caso contrário deve-se utilizar os índices temporários;
- Para gerar os arquivos DFF's compatíveis com outras ferramentas que manipulam arquivos btrieve, inclusive Crystal Reports, existem duas funções para criar os arquivos necessários: <@>BTVTables e <@>BTVCreateDDFs;

Configuração do Btrieve/PervasiveSQL2000:

- Para configurar o Btrieve Server deve-se executar os seguintes passos:
 1. Terminar a execução do Protheus e parar o serviço w32mkde;
 2. Deletar os arquivos binários do Btrieve do diretório "Protheus\Bin" (wbtrv32.dll, w32mkrc.dll, wbtrvres.dll e w32mkde.exe);
 3. Instalar o PervasiveSQL200 com os respectivos "Services Packs";
 4. Entrar no Pervasive Control Center (menu Iniciar - Pervasive - Pervasive Control Center);
 5. Visualizar os nomes dos servidores disponíveis (no caso de Linux, Novell e Solaris deve-se acrescentar um servidor);
 6. Através de um duplo click sobre o servidor que se deseja utilizar entrar em "configuração" (configuration);
 7. Para Windows NT e 2000 deve-se acertar os valores de alguns parâmetros:

Pasta Access:

 - Accept Remote Request : ON
 - Active Clients : 10000
 - Logical File Handles : 100000
 - MaxDatabases: 10
 - Maximum Open Files: 10000
 - Number of Sessions: 20

Pasta Communication Buffer Size:

- Communication Buffer Size : 63
- MKDE Communication Buffer Size: 63
- Read Buffer Size: 4

Pasta Data Integrity:

- 10. Initiation Time Limit: 100000
- 11. Operation Bundle Limit: 10000

Pasta Memory usage:

- Allocate Resource at Startup: On
- Back to Minimal State if Inactive: On
- Extended Operation Buffer Size: 16
- System Cache: On

Pasta Performance Tunning:

- Cache Allocation Size: +- 131072 (mínimo de 12000)
- Communications Threads : 64
- Index Balancing: Off
- Largest Compressed Record Size: 0
- Log Buffer Size: 64
- Number of Input/ Output Threads : 64
- Number of Worker Threads: 64
- Transaction Log Size: 512

*Obs: O Cache Allocation Size aloca memória do servidor para uso do banco de dados. Quanto mais memória, mais rápidas são executadas as operações.

•

A versão 6.15 possui uma limitação: Se dois servidores NT4 estiverem acessando o mesmo arquivo, ocorrerá lentidão na rede. Isso acontece se for utilizado o mesmo RPO (que é um arquivo btrieve) para dois servidores no AP5. A solução é sempre replicar os repositórios em cada servidor ou adquirir (deve ser comprado) a versão a partir da PervasiveSQL2000. Em máquinas Win2000, não é nem mesmo possível abrir um arquivo btrieve de dois servidores. Este problema é de conhecimento da Pervasive, mas não será alterado porque esta versão foi descontinuada por volta de agosto de 2001.

Ctree

Informações Básicas

- Para manipulação de tabelas Ctree o driver utilizado é "CTREECDX";
- Para programar algo específico para o Ctree pode-se utilizar o teste "#ifdef CTREE";
- A extensão padrão das tabelas é ".dtc". Quando o LocalFile estiver utilizando o banco Ctree os SXs continuam tendo como padrão a extensão ".dbf", mas as tabelas criadas (SX1990.DBF, SX2990.DBF, etc) são Ctree. Portanto recomenda-se que se configure outra extensão padrão para arquivos locais do tipo Ctree através da chave "LocalDbExtension" no arquivo "apósrv.ini" como ".dtc";
- As manipulações e visualizações de dados devem ser feitas através do "APSdu", pois as antigas ferramentas são incompatíveis;
- O Ctree não precisa ser instalado, pois sua biblioteca é gerada junto com o Protheus;
- Os campos do tipo memo devem ser os últimos da tabela, por isso na hora da sua criação o Protheus automaticamente desloca-os para o final e mostra uma mensagem de aviso;
- As tabelas geradas pelo Ctree são totalmente compatíveis entre as plataformas Windows e Linux, pode-se inclusive copiar uma tabela gerada no Linux e abri-la no Windows e vice-e-versa.

Índices

- - O índice interno do Ctree (ordem do recno) é criado em outro arquivo com extensão ".int". Uma pasta ("ctreeint") é criada abaixo da pasta com a tabela. Nesta pasta serão armazenados todos os índices internos daquele diretório. Caso não exista o arquivo de índice interno o mesmo é gerado automaticamente pelo Protheus mostrando um aviso de que reconstruiu o índice no servidor na hora da abertura da tabela. Para apagá-lo pode ser utilizada uma função <@>CTREEDELINT;
- - Os arquivos de índices permanentes são criados fora do arquivo da tabela com extensão padrão como nos outros RDDs (".cdx"), mas suas informações são armazenadas no arquivo da tabela (".dtc"). Portanto para se criar ou excluir índices permanentes a tabela deve estar aberta em modo exclusivo. Na hora da abertura da tabela, todos os arquivos de índices permanentes relacionados em sua estrutura são abertos também, por isso não se pode deletar o arquivo de índice permanente com a respectiva tabela aberta. Caso não exista um ou mais arquivos de índices da tabela na hora de sua abertura, o Protheus irá recriá-los automaticamente de forma semelhante ao índice interno. O diretório do arquivo de índice também é armazenado na estrutura da tabela, mas quando a tabela é aberta e é constatado que a tabela está em outro diretório o Protheus automaticamente atualiza esta informação. Para se deletar os índices de uma tabela Ctree pode-se utilizar a função <@>CTREEDELIDX ou utilizar a opção "Index/erase all" no APSdu. Portanto aconselha-se utilizar o índice do tipo permanente somente se o mesmo for utilizado posteriormente (outras aberturas da tabela) caso contrário deve-se utilizar os índices temporários;
- - O índice temporário é criado dentro de um subdiretório com o nome do arquivo especificado na hora de sua criação, por exemplo "ind1.idx" contendo os arquivos "ind1.ind", "ind1c.ind" e "ind1r.ind". Este tipo de índice não possui definição armazenada no arquivo da tabela, por ser temporário.

Sistemas Operacionais e Plataformas (versão AP6 e posteriores)

Sistemas Operacionais e Plataformas

O Protheus Server foi desenvolvido em ANSI C++ e , portanto, independe de API's específicas para funcionar. Graças a isso, o núcleo do Protheus pode ser recompilado em todos os sistemas operacionais e plataformas que suportem ANSI C++.

Outra preocupação durante o desenvolvimento do Protheus foi garantir total compatibilidade dos repositórios de objetos do Protheus (RPO's) e das correções dos repositórios (Patch's) entre os sistemas operacionais e plataformas.

Plataformas e SO's suportados

- Windows e Linux Intel,
- Windows IA64,
- Sun Solaris (RISC),
- HP UX (RISC),
- Compaq True64 (RISC),
- IBM AIX (Power PC e RS/6000),

Novos SO's a serem suportados

- PalmOS (em fase Beta)
- PocketPC

AdPI – Guia de Programação

A Linguagem AdvPI

A Linguagem AdvPI 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 AdvPI 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 AP6 Server para a execução. Como não existe a linkedição, ou união física do código compilado a um determinado módulo ou aplicação, funções criadas em AdvPI podem ser executadas em qualquer ponto do ambiente Advanced Protheus.

O compilador e o interpretador da linguagem AdvPI é o próprio servidor AP6 (AP6 Server), e existe um ambiente visual para desenvolvimento integrado (AP6 IDE) onde o código pode ser criado, compilado e depurado.

Os programas em AdvPI 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 AP6 Remote. O AP6 Remote é a aplicação encarregada da interface e da interação com o usuário, sendo que todo o processamento do código em AdvPI, o acesso ao banco de dados e o gerenciamento de conexões é efetuado no AP6 Server. O AP6 Remote é o principal meio de acesso a execução de rotinas escritas em AdvPI no AP6 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.

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

Todo o código do sistema ERP Advanced Protheus é escrito em AdvPI.

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 AP6 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 AP6 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 AP6 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 AdvPI podem ser iniciadas como processos individuais (sem interface) no AP6 Server através de duas maneiras: Iniciadas por outra rotina AdvPI através da chamada de funções como StartJob ou CallProc ou iniciadas automaticamente na inicialização do AP6 Server (quando propriamente configurado).

Programação de RPC

Através de uma biblioteca de funções disponível no Protheus (uma API de comunicação), pode-se executar rotinas escritas em AdvPI diretamente no AP6 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 AdvPI 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 AdvPI através de conexão TCP/IP direta.

Programação Web

O AP6 Server pode também ser executado como um servidor Web, respondendo a requisições HTTP. No momento destas requisições, pode executar rotinas escritas em AdvPI 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 AdvPI que não contenha comandos de interface pode ser executada através de requisições HTTP. O Protheus permite a compilação de arquivos HTML contendo código AdvPI embutido. São os chamados arquivos AdvPI 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 AP6 Server pode emular um terminal TelNet, através da execução de rotinas escritas em AdvPI. Ou seja, pode-se escrever rotinas AdvPI cuja interface final será um terminal TelNet ou um coletor de dados móvel.

Sintaxe da Linguagem

Edição de Programas

Criação de um Programa

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 AdvPI, não é o sistema operacional de um computador que irá executar o código compilado, mas sim o AP6 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 a 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 AdvPI 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 AdvPI também permite que existam linhas de comando com comentário. Isto é possível incluindo-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.

Estrutura de um Programa

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:

```
/*
=====+0
| Programa: Cálculo do Fatorial          |
| Autor   : Microsiga Software S.A.     |
| Data    : 02 de outubro de 2001      |
=====+
*/

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
```

Pode-se classificar um programa em AdvPI em quatro partes básicas:

1. Área de Identificação
2. Área de Ajustes Iniciais
3. Corpo do Programa
4. Área de Encerramento

A 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.

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

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

A 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 AdvPI 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
```

O 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 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 AdvPI 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
```

Variáveis de Memória

Tipos de Dados

O AdvPI não é uma linguagem de tipos rígidos (strongly typed), o que significa que variáveis de memória podem diferentes tipos de dados durante a execução do programa. Variáveis podem também conter objetos, mas os tipos primários da linguagem são:

Numérico

O AdvPI não diferencia valores inteiros de valores com ponto flutuante, portanto pode-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 AdvPI são identificados através de .T. ou .Y. para verdadeiro e .F. ou .N. para falso (independentemente se os caracteres estiverem em maiúsculo ou minúsculo).

Caracter

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

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

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

Data

O AdvPI 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.

Matriz (Array)

Matrizes são um tipo de dado especial. É a disposição de outros elementos em colunas e linhas. O AdvPI suporta matrizes uni ou multidimensionais. Os elementos de uma matriz são acessados através de índices numéricos iniciados em 1, identificando a linha e coluna para quantas dimensões existirem.

Uma matriz pode conter no máximo 100000 elementos, independentemente do número de dimensões.

Matrizes 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 AdvPI que poderão ser executadas posteriormente.

Criação e Atribuição 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 que segue duas regras:

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.

Limitação de caracteres no nome. Os nomes das variáveis devem sempre começar por uma letra ou o caracter de sublinhado (_). No restante, pode conter letras, números e o caracter de sublinhado. Qualquer outro caracter, incluindo espaços em branco, não são permitidos.

O AdvPl permite a criação ilimitada de variáveis, dependendo apenas da memória disponível. A seguir estão alguns nomes válidos para variáveis:

```
TOT01
cNumero
VAR_QUALQUER
M_CARGO
All
```

E alguns inválidos:

```
1CODIGO (Inicia por um número)
M CARGO (contém um espaço em branco)
LOCAL   (palavra reservada do AdvPl)
```

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*, seguido de uma lista de variáveis separadas por vírgula (,). 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. Os diferentes tipos de contexto de variáveis são explicados na documentação sobre [escopo de 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  
Store 10 To nPercentual  
nResultado := 250 * (1 + (nPercentual / 100))
```

ou

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

ou

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

A diferença entre o último exemplo e os dois anteriores é que a variável é inicializada no momento da declaração. Nos dois primeiros exemplos, a variável é primeiro declarada e então inicializada em uma outra linha de código. O comando store existe apenas por compatibilidade com versões anteriores e outras linguagens xBase, mas é obsoleto. Deve-se utilizar o operador de atribuição (:= ou somente =). 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.

Uma vez que um valor lhe seja atribuído, o tipo de dado de uma variável é igual ao tipo de dado do valor atribuído. Ou seja, uma variável passa a ser numérica se um número lhe é atribuído, passa a ser caracter se uma string de texto lhe for atribuída, etc. Porém mesmo que uma variável seja de determinado tipo de dado, pode-se mudar o tipo da variável atribuindo outro tipo a ela:

```
01 Local xVariavel // Declara a variável inicialmente com valor nulo
```

```
02
03 xVariavel := "Agora a variável é caracter..."
04 Alert("Valor do Texto: " + xVariavel)
05
06 xVariavel := 22 // Agora a variável é numérica
07 Alert(cValToChar(xVariavel))
08
09 xVariavel := .T. // Agora a variável é lógica
10 If xVariavel
11     Alert("A variável tem valor verdadeiro...")
12 Else
13     Alert("A variável tem valor falso...")
14 Endif
15
16 xVariavel := Date() // Agora a variável é data
17 Alert("Hoje é: " + DtoC(xVariavel))
18
19 xVariavel := nil // Nulo novamente
20 Alert("Valor nulo: " + xVariavel)
21
22 Return
```

No programa de exemplo anterior, a variável `xVariavel` é utilizada para armazenar diversos tipos de dados. A letra "x" em minúsculo no começo do nome é utilizada para indicar uma variável que pode conter diversos tipos de dados, segundo a Notação Húngara (consulte documentação específica para detalhes). Este programa troca os valores da variável e exibe seu conteúdo para o usuário através da função `Alert`. Essa função recebe um parâmetro que deve ser do tipo string de caracter, por isso dependendo do tipo de dado da variável `xVariavel` é necessário fazer uma conversão antes.

Apesar dessa flexibilidade de utilização de variáveis, deve-se tomar cuidados na passagem de parâmetros para funções ou comandos, e na concatenação (ou soma) de valores. Note a linha 20 do programa de exemplo. Quando esta linha é executada, a variável `xVariavel` contém o valor nulo. A tentativa de soma de tipos de dados diferentes gera erro de execução do programa. Nesta linha do exemplo, ocorrerá um erro com a mensagem "type mismatch on +". Excetuando-se o caso do valor nulo, para os demais deve-se sempre utilizar funções de conversão quando necessita-se concatenar tipos de dados diferentes (por exemplo, nas linhas 07 e 17).

Note também que quando uma variável é do tipo de dado lógico, ela pode ser utilizada diretamente para checagem (linha 10):

If `xVariavel`

é o mesmo que

If `xVariavel = .T.`

A declaração de variáveis para os demais tipos de dados, matrizes e blocos de código, é exatamente igual ao descrito até agora. Apenas existem algumas diferenças quanto a inicialização, que podem ser consultadas na documentação de [inicialização de matrizes](#) e [blocos de código](#).

Matrizes

Matrizes, ou arrays, são coleções de valores. Ou, de uma maneira mais fácil de entender, uma lista. Uma matriz pode ser criada através de diferentes maneiras. Consulte a documentação sobre [Inicialização de Matrizes](#) para maiores detalhes.

Cada item em uma matriz é referenciado pela indicação de sua posição numérica na lista, iniciando pelo número 1. O exemplo a seguir declara uma variável, atribui uma matriz de três elementos a ela, e então exibe um dos elementos e o tamanho da matriz:

```
Local aLetras          // Declaração da variável

aLetras := { "A", "B", "C" } // Atribuição da matriz à variável

Alert(aLetras[2])      // Exibe o segundo elemento da matriz

Alert(cValToChar(Len(aLetras))) // Exibe o tamanho da matriz
```

O AdvPI permite a manipulação de matrizes facilmente. Enquanto que em outras linguagens como C ou Pascal é necessário alocar memória para cada elemento de uma matriz (o que tornaria a utilização de "pointeiros" necessária), o AdvPI se encarrega de gerenciar a memória e torna simples adicionar elementos a uma matriz, utilizando a função `aAdd`:

```
aAdd(aLetras, "D") // Adiciona o quarto elemento ao final da matriz

Alert(aLetras[4]) // Exibe o quarto elemento

Alert(aLetras[5]) // Erro! Não há um quinto elemento na matriz
```

Matrizes como Estruturas

Uma característica interessante do AdvPI é que uma matriz pode conter qualquer coisa: números, datas, lógicos, caracteres, objetos, etc. E ao mesmo tempo. Em outras palavras, os elementos de uma matriz não precisam ser necessariamente do mesmo tipo de dado, em contraste com outras linguagens como C e Pascal.

```
aFunc1 := { "Pedro", 32, .T. }
```

Esta matriz contém uma string, um número e um valor lógico. Em outras linguagens como C ou Pascal, este "pacote" de informações pode ser chamado como um "struct" (estrutura em C, por exemplo) ou um "record" (registro em Pascal, por exemplo). Como se fosse na verdade um registro de um banco de dados, um pacote de informações construído com diversos campos. Cada campo tendo um pedaço diferente de dado.

Suponha que no exemplo anterior, o array `aFunc1` contenha informações sobre o nome de uma pessoa, sua idade e sua situação matrimonial. Os seguintes `#define`s podem ser criados para indicar cada posição dos valores dentro da matriz:

```
#define FUNCT_NOME 1

#define FUNCT_IDADE 2

#define FUNCT_CASADO 3
```

E considere mais algumas matrizes para representar mais pessoas:

```
aFunc2 := {"Maria" , 22, .T.}
```

```
aFunc3 := {"Antônio", 42, .F.}
```

Os nomes podem ser impressos assim:

```
Alert(aFunc1[FUNCT_NOME])
```

```
Alert(aFunc2[FUNCT_NOME])
```

```
Alert(aFunc3[FUNCT_NOME])
```

Agora, ao invés de trabalhar com variáveis individuais, pode-se agrupá-las em uma outra matriz, do mesmo modo que muitos registros são agrupados em uma tabela de banco de dados:

```
aFuncs := {aFunc1, aFunc2, aFunc3}
```

Que é equivalente a isso:

```
aFuncs := { {"Pedro" , 32, .T.}, ;
```

```
          {"Maria" , 22, .T.}, ;
```

```
          {"Antônio", 42, .F.} }
```

aFuncs é uma matriz com 3 linhas por 3 colunas. Uma vez que as variáveis separadas foram combinadas em uma matriz, os nomes podem ser exibidos assim:

```
Local nCount
```

```
For nCount := 1 To Len(aFuncs)
```

```
    Alert(aFuncs[nCount,FUNCT_NOME])
```

```
        // O acesso a elementos de uma matriz multidimensional
```

```
        // pode ser realizado também desta forma:
```

```
    // aFuncs[nCount][FUNCT_NOME]
```

```
Next nCount
```


A variável nCount seleciona que funcionário (ou que linha) é de interesse. Então a constante FUNCT_NOME seleciona a primeira coluna daquela linha.

Cuidados com Matrizes

Matrizes são listas de elementos, portanto memória é necessária para armazenar estas informações. Como as matrizes podem ser multidimensionais, a memória necessária será a multiplicação do número de itens em cada dimensão da matriz, considerando-se o tamanho do conteúdo de cada elemento contido nesta. Portanto o tamanho de uma matriz pode variar muito.

A facilidade da utilização de matrizes, mesmo que para armazenar informações em pacotes como descrito anteriormente, não é compensada pela utilização em memória quando o número de itens em um array for muito grande. Quando o número de elementos for muito grande deve-se procurar outras soluções, como a utilização de um arquivo de banco de dados temporário.

Não há limitação para o número de dimensões que uma matriz pode ter, mas o número de elementos máximo (independentes das dimensões onde se encontram) é de 100000.

Iniciando Matrizes

Algumas vezes o tamanho da matriz é conhecido previamente. Outras vezes o tamanho da matriz só será conhecido em tempo de execução.

Se o tamanho da matriz é conhecido

Se o tamanho da matriz é conhecido no momento que o programa é escrito, há diversas maneiras de implementar o código.

```
01 Local nCnt
02 Local aX[10]
03 Local aY := Array(10)
04 Local aZ := {0,0,0,0,0,0,0,0,0,0}
05
06 For nCnt := 1 To 10
07   aX[nCnt] := nCnt * nCnt
08 Next nCnt
```

Este código preenche a matriz com uma tabela de quadrados. Os valores serão 1, 4, 9, 16 ... 81, 100. Note que a linha 07 se refere à variável aX, mas poderia também trabalhar com aY ou aZ. O objetivo deste exemplo é demonstrar três modos de criar uma matriz de tamanho conhecido no momento da criação do código.

Na linha 02 a matriz é criada usando aX[10]. Isto indica ao AdvPI para alocar espaço para 10 elementos na matriz. Os colchetes [e] são utilizados para indicar o tamanho necessário.

Na linha 03 é utilizada a função array com o parâmetro 10 para criar a matriz, e o retorno desta função é atribuído à variável aY.

Na linha 03 é efetuado o que se chama "desenhar a imagen da matriz". Como pode-se notar, existem dez 0's na lista encerrada entre chaves ({}). Claramente, este método não é o utilizado para criar uma matriz de 1000 elementos. O terceiro método difere dos anteriores porque inicializa a matriz com os valores definitivos. Nos dois primeiros métodos, cada posição da matriz contém um valor nulo (Nil) e deve ser inicializado posteriormente.

A linha 07 demonstra como um valor pode ser atribuído para uma posição existente em uma matriz especificando o índice entre colchetes.

Se o tamanho da matriz não é conhecido

Se o tamanho da matriz não é conhecido até o momento da execução do programa, há algumas maneiras de criar uma matriz e adicionar elementos a ela. O exemplo a seguir ilustra a idéia de criação de uma matriz vazia (sem nenhum elemento) e adição de elementos dinamicamente.

```
01 Local nCnt
02 Local aX[0]
03 Local aY := Array(0)
04 Local aZ := {}
05
06 For nCnt := 1 To nSize
07   aAdd(aX,nCnt*nCnt)
08 Next nCnt
```

A linha 02 utiliza os colchetes para criar uma matriz vazia. Apesar de não ter nenhum elemento, seu tipo de dado é matriz.

Na linha 03 a chamada da função array cria uma matriz sem nenhum elemento.

Na linha 04 está declarada a representação de uma matriz vazia em AdvPl. Mais uma vez, estão sendo utilizadas as chaves para indicar que o tipo de dados da variável é matriz. Note que {} é uma matriz vazia (tem o tamanho 0), enquanto { Nil } é uma matriz com um único elemento nulo (tem tamanho 1).

Porque cada uma destas matrizes não contem elementos, a linha 07 utiliza a função aadd para adicionar elementos sucessivamente até o tamanho necessário (especificado por exemplo na variável nSize).

Blocos de Código

Blocos de código são um conceito existente há muito tempo em linguagens xBase. Não como algo que apareceu da noite para o dia, e sim uma evolução progressiva utilizando a combinação de muitos conceitos da linguagem para a sua implementação.

Um Primeiro Lembrete

O AdvPl é uma linguagem baseada em funções. Funções têm um valor de retorno. Assim como o operador de atribuição :=.

Assim, ao invés de escrever:

```
x := 10 // Atribui o valor 10 à variável chamada X  
Alert("Valor de x: " + cValToChar(x))
```

Pode-se escrever:

```
// Atribui e então exibe o valor da variável X  
Alert("Valor de x: " + cValtoChar(X := 10))
```

A expressão `x:=10` é avaliada primeiro, e então seu resultado (o valor de X, que agora é 10) é passada para a função `cvaltochar` para a conversão para caracter, e em seguida para a função `alert` para a exibição. Por causa desta regra de precedência é possível atribuir um valor a mais de uma variável ao mesmo tempo:

```
Z := Y := X := 0
```

Por causa dessa regra, essa expressão é avaliada como se fosse escrita assim:

```
Z := ( Y := ( X := 0 ) )
```

Apesar do AdvPl avaliar expressões da esquerda para a direita, no caso de atribuições isso acontece ao contrário, da direita para a esquerda. O valor é atribuído à variável X, que retorna o valor para ser atribuído à variável Y e assim sucessivamente. Pode-se dizer que o zero foi "propagado através da expressão".

Outro Lembrete

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

```
If IAchou  
    Alert("Cliente encontrado!")  
Endif
```

pode ser escrito assim:

```
If IAchou ; Alert("Cliente encontrado!") ; Endif
```

O ponto-e-vírgula indica ao AdvPI que a nova linha de código está para começar. Pode-se então colocar diversas linhas lógicas de código na mesma linha física através do editor de texto utilizado.

Apesar da possibilidade de se escrever todo o programa assim, em uma única linha física, isto não é recomendado pois dificulta a legibilidade do programa e, conseqüentemente, a manutenção.

Lista de Expressões

A evolução dos blocos de código começa com as listas de expressões. Nos exemplos a seguir, o símbolo `==>` indicará o retorno da expressão após sua avaliação (seja para atribuir em uma variável, exibir para o usuário ou imprimir em um relatório), que será impresso em um relatório por exemplo.

Duas Linhas de Código

```
@00,00 PSAY x := 10    ==>    10
```

```
@00,00 PSAY y := 20    ==>    20
```

Cada uma das linhas terá a expressão avaliada, e o valor da variável será então impresso.

Duas Linha de Código em Uma , Utilizando Ponto-e-Vírgula

Este é o mesmo código que o anterior, apenas escrito em uma única linha:

```
Alert( cValToChar( x := 10 ; y := 20 ) )    ==>    10
```

Apesar desse código se encontrar em uma única linha física, existem duas linhas lógicas separadas pelo ponto e vírgula. Ou seja, esse código é equivalente a:

```
Alert( cValToChar( x := 10 ) )
```

```
y := 20
```

Portanto apenas o valor 10 da variável x será passado para as funções `cvaltochar` e `alert` para ser exibido. E o valor 20 apenas será atribuído à variável y.

Convertendo para uma Lista de Expressões

Quando parênteses são colocados ao redor do código e o sinal de ponto-e-vírgula substituído por uma vírgula apenas, o código torna-se uma lista de expressões:

```
Alert( cValToChar ( ( X := 10 , Y := 20 ) ) )    ==>    20
```

O valor de retorno resultante de uma lista de expressões é o valor resultante da última expressão ou elemento da lista. Funciona como se fosse um pequeno programa ou função, que retorna o resultado de sua última avaliação (efetuadas da esquerda para a direita).

Neste exemplo, a expressão `x := 10` é avaliada, e então a expressão `y := 20`, cujo valor resultante é passado para a função `alert` e `cvaltochar`, e então exibido. Depois que essa linha de código é executada, o valor de `X` é igual a 10 e o de `y` igual a 20, e 20 será exibido.

Teoricamente, não há limitação para o número de expressões que podem ser combinadas em uma lista de expressões. Na prática, o número máximo é por volta de 500 símbolos.

Debugar listas de expressões é difícil porque as expressões não estão divididas em linhas de código fonte, o que torna todas as expressões associadas a uma mesma linha de código. Isto pode tornar muito difícil determinar onde um erro ocorreu.

Onde Pode-se Utilizar uma Lista de Expressões?

O propósito principal de uma lista de expressões é agrupá-las em uma única unidade. Em qualquer lugar do código AdvPI que uma expressão simples pode ser utilizada, pode-se utilizar uma lista de expressões. E ainda, pode-se fazer com que várias coisas aconteçam onde normalmente apenas uma aconteceria.

```
X := 10 ; Y := 20
```

```
If X > Y
```

```
    Alert("X")
```

```
    Z := 1
```

```
Else
```

```
    Alert("Y")
```

```
    Z := -1
```

```
Endif
```

Aqui temos o mesmo conceito, escrito utilizando listas de expressões na função `iif`:

```
X := 10 ; Y := 20
```

```
iif( X > Y , ;
```

```
    ( Alert("X"), Z := 1 ) , ;
```

```
    ( Alert("Y"), Z := -1 ) )
```

De Listas de Expressões para Blocos de Código

Considere a seguinte lista de expressões:

```
Alert( cValToChar( ( x := 10, y := 20 ) ) ) ==> 20
```

O AdvPI permite criar funções, que são pequenos pedaços de código, como se fosse um pequeno programa, utilizados para diminuir partes de tarefas mais complexas e reaproveitar código em mais de um lugar num programa. Para maiores detalhes consulte a documentação sobre a criação de funções em AdvPI. Porém, a idéia neste momento é que a lista de expressões utilizada na linha anterior pode ser criada como uma função:

```
Function Lista()
```

```
X := 10
```

```
Y := 20
```

```
Return Y
```

E a linha de exemplo com a lista de expressões pode ser substituída, tendo o mesmo resultado, por:

```
Alert( cValToChar( Lista() ) ) ==> 20
```

Como mencionado anteriormente, uma lista de expressões é como um pequeno programa ou função. Com poucas mudanças, uma lista de expressões pode se tornar um bloco de código:

```
( X := 10 , Y := 20 ) // Lista de Expressões
```

```
{ || X := 10 , Y := 20 } // Bloco de Código
```

Note as chaves {} utilizadas no bloco de código. Ou seja, um bloco de código é uma matriz. Porém na verdade, não é uma lista de dados, e sim uma lista de comandos, uma lista de código.

```
// Isto é uma matriz de dados
```

```
A := { 10, 20, 30 }
```

```
// Isto é um bloco de código, porém funciona como
```

```
// se fosse uma matriz de comandos
```

```
B := { || x := 10, y := 20 }
```

Executando um Bloco de Código

Diferentemente de uma matriz, não se pode acessar elementos de um bloco de código através de um índice numérico. Porém blocos de código são semelhantes a uma lista de expressões, e a uma pequena função. Ou seja, podem ser executados. Para a execução, ou avaliação, de um bloco de código, deve-se utilizar a função eval:

```
nRes := Eval(B) ==> 20
```

Essa função recebe como parâmetro um bloco de código e avalia todas as expressões contidas neste bloco de código, retornando o resultado da última expressão avaliada.

Passando Parâmetros

Já que blocos de código são como pequenas funções, também é possível a passagem de parâmetros para um bloco de código. Os parâmetros devem ser informados entre as barras verticais (|) separados por vírgulas, assim como em uma função.

```
B := { | N | X := 10, Y := 20 + N }
```

Porém deve-se notar que já que o bloco de código recebe um parâmetro, um valor deve ser passado quando o bloco de código for avaliado.

```
C := Eval(B, 1) ==> 21
```

Utilizando Blocos de Código

Blocos de código podem ser utilizados em diversas situações. Geralmente são utilizados para executar tarefas quando eventos de objetos são acionados ou para modificar o comportamento padrão de algumas funções.

Por exemplo, considere a matriz abaixo:

```
A := { "GARY HALL", "FRED SMITH", "TIM JONES" }
```

Esta matriz pode ser ordenada pelo primeiro nome, utilizando-se a chamada da função `asort(A)`, resultado na matriz com os elementos ordenados dessa forma:

```
{ "FRED SMITH", "GARY HALL", "TIM JONES" }
```

A ordem padrão para a função `asort` é ascendente. Este comportamento pode ser modificado através da informação de um bloco de código que ordena a matriz de forma descendente:

```
B := { |X, Y| X > Y }
```

```
aSort(A, B)
```

O bloco de código (de acordo com a documentação da função `asort`) deve ser escrito para aceitar dois parâmetros que são os dois elementos da matriz para comparação. Note que o bloco de código não conhece que elementos está comparando - a função `asort` seleciona os elementos (talvez utilizando o algoritmo *QuickSort*) e passa-os para o bloco de código. O bloco de código compara-os e retorna verdadeiro (.T.) se se encontram na ordem correta, ou falso (.F.) se não. Se o valor de retorno for falso, a função `asort` irá então trocar os valores de lugar e seguir comparando o próximo par de valores.

Então, no bloco de código anterior, a comparação $X > Y$ é verdadeira se os elementos estão em ordem decrescente, o que significa que o primeiro valor é maior que o segundo.

Para ordenar a mesma matriz pelo último nome, também em ordem decrescente, pode-se utilizar o seguinte bloco de código:

```
B := { |X, Y| Substr(X,At(" ",X)+1) > Substr(Y,At(" ",Y)+1) }
```

Note que este bloco de código procura e compara as partes dos caracteres imediatamente seguinte a um espaço em branco. Depois de utilizar esse bloco de código para a função `asort`, a matriz conterá:

```
{"GARY HALL", "TIM JONES", "FRED SMITH"}
```

Finalmente, para ordenar um sub-elemento (coluna) de uma matriz por exemplo, pode-se utilizar o seguinte bloco de código:

```
B := { |X, Y| X[1] > Y[1] }
```

Escopo de Variáveis

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 AdvPI 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 AdvPI 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 pretende-se fazer uma atribuição a uma variável declarada previamente mas escreve-se o nome da variável de forma incorreta, o AdvPI 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 Locais

Variáveis locais são pertencentes apenas ao escopo da função onde foram declaradas. 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 locais 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 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 locais criadas na mais recente ativação são visíveis.

Variáveis Estáticas

Variáveis estáticas funcionam basicamente como as variáveis locais, mas mantêm seu valor através da execução. Variáveis estáticas devem ser declaradas explicitamente no código com o identificador `STATIC`.

O escopo das variáveis estáticas 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 é todo o arquivo de programa.

Neste exemplo, a variável `nVar` é declarada como estática 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 Privadas

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 como privada e inicializada com o valor 10. Quando a função Filha é executada, nVar ainda existe e, diferente de uma variável 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.

Variáveis Públicas

Pode-se criar variáveis públicas dinamicamente no código com o identificador PUBLIC. As variáveis públicas continuam a existir e mantêm seu valor até o fim da execução.

É possível criar uma variável privada com o mesmo nome de uma variável pública existente. Entretanto, não é permitido criar uma variável pública com o mesmo nome de uma variável privada existente.

Uma vez criada, uma variável pública é visível em todo o programa onde foi declarada até que seja *escondida* por uma variável privada criada com o mesmo nome. A nova variável privada criada *esconde* a

variável pública existente, e esta se tornará inacessível até que a nova variável privada seja destruída. Por exemplo:

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

Neste exemplo, nVar é criada como pública 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).

Diferenciação entre variáveis e nomes de campos

Muitas vezes uma variável pode ter o mesmo nome que um campo de um arquivo ou tabela aberto no momento. Neste caso, o AdvPL privilegiará o campo. Assim uma referência a um nome que identifique tanto uma variável como um campo, resultará no conteúdo do campo.

Para especificar qual deve ser o elemento referenciado, deve-se utilizar o operador de identificação de apelido (->) e um dos dois identificadores de referência, MEMVAR ou FIELD.

```
cRes := MEMVAR->NOME
```

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

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

Neste caso, o valor atribuído à variável cRes será o valor do campo NOME existente no arquivo ou tabela aberto na área atual.

O identificador FIELD pode ser substituído pelo apelido de um arquivo ou tabela aberto, para evitar a necessidade de selecionar a área antes de acessar o conteúdo de terminado campo.

```
cRes := CLIENTES->NOME
```

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

Operadores da Linguagem

Operadores Comuns

Na documentação sobre variáveis há uma breve demonstração de como atribuir valores a uma variável da forma mais simples. O AdvPI amplia significativamente a utilização de variáveis através do uso de expressões e funções. Uma expressão é um conjunto de operadores e operandos cujo resultado pode ser atribuído a uma variável ou então analisado para a tomada de decisões. Por exemplo:

```
Local nSalario := 1000, nDesconto := 0.10
```

```
Local nAumento, nSalLiquido
```

```
nAumento := nSalario * 1.20
```

```
nSalLiquido := nAumento * (1-nDesconto)
```

Neste exemplo são utilizadas algumas expressões para calcular o salário líquido após um aumento. Os operandos de uma expressão podem ser uma variável, uma constante, um campo de arquivo ou uma função.

Operadores Matemáticos

Os operadores utilizados em AdvPI para cálculos matemáticos são:

+	Adição
-	Subtração
*	Multiplicação

/	Divisão
** ou ^	Exponenciação
%	Módulo (Resto da Divisão)

Operadores de String

Os operadores utilizados em AdvPI para tratamento de caracteres são:

+	Concatenação de strings (união)
-	Concatenação de strings com eliminação dos brancos finais das strings intermediárias
\$	Comparação de Substrings (contido em)

Operadores Relacionais

Os operadores utilizados em AdvPI para operações e avaliações relacionais são:

<	Comparação Menor
>	Comparação Maior
=	Comparação Igual
==	Comparação Exatamente Igual (para caracteres)
<=	Comparação Menor ou Igual
>=	Comparação Maior ou Igual
<> ou # ou !=	Comparação Diferente

Operadores Lógicos

Os operadores utilizados em AdvPI para operações e avaliações lógicas são:

.And.	E lógico
.Or.	OU lógico

.Not. ou !	NÃO lógico
------------	------------

Operadores de Atribuição

Os operadores utilizados em AdvPI para atribuição de valores a variáveis de memória são:

=	Atribuição Simples
:=	Atribuição em Linha
+=	Adição e Atribuição em Linha
-=	Subtração e Atribuição em Linha
*=	Multiplicação e Atribuição em Linha
/=	Divisão e Atribuição em Linha
** = ou ^=	Exponenciação e Atribuição em Linha
%=	Módulo (resto da divisão) e Atribuição em Linha

Atribuição Simples

O sinal de igualdade é utilizado para atribuir valor a uma variável de memória.

nVariavel = 10

Atribuição em Linha

O operador de atribuição em linha é caracterizado por dois pontos e o sinal de igualdade. Tem a mesma função do sinal de igualdade sozinho, porém aplica a atribuição às variáveis. Com ele pode-se atribuir mais de uma variável ao mesmo tempo.

nVar1 := nVar2 := nVar3 := 0

Quando diversas variáveis são inicializadas em uma mesma linha, a atribuição começa da direita para a esquerda, ou seja, nVar3 recebe o valor zero inicialmente, nVar2 recebe o conteúdo de nVar3 e nVar1 recebe o conteúdo de nVar2 por final.

Com o operador de atribuição em linha, pode-se substituir as inicializações individuais de cada variável por uma inicialização apenas:

```
Local nVar1 := 0, nVar2 := 0, nVar3 := 0
```

por

```
Local nVar1 := nVar2 := nVar3 := 0
```

O operador de atribuição em linha também pode ser utilizado para substituir valores de campos em um banco de dados.

Atribuição Composta

Os operadores de atribuição composta são uma facilidade da linguagem AdvPI para expressões de cálculo e atribuição. Com eles pode-se economizar digitação:

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

Operadores de Incremento/Decremento

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

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

Os operadores de decremento/incremento podem ser colocados tanto antes (pré-fixado) como depois (pós-fixado) do nome da variável. Dentro de uma expressão, a ordem do operador é muito importante, podendo alterar o resultado da expressão. Os operadores incrementais são executados da esquerda para a direita dentro de uma expressão.

```
Local nA := 10
```



```
Local nB := nA++ + nA
```

O valor da variável nB resulta em 21, pois a primeira referência a nA (antes do ++) continha o valor 10 que foi considerado e imediatamente aumentado em 1. Na segunda referência a nA, este já possuía o valor 11. O que foi efetuado foi a soma de 10 mais 11, igual a 21. O resultado final após a execução destas duas linhas é a variável nB contendo 21 e a variável nA contendo 11.

No entanto:

```
Local nA := 10
```

```
Local nB := ++nA + nA
```

Resulta em 22, pois o operador incremental aumentou o valor da primeira nA antes que seu valor fosse considerado.

Operadores Especiais

Além dos operadores comuns, o AdvPI possui alguns outros operadores ou identificadores. Estas são suas finalidades:

()	Agrupamento ou Função
[]	Elemento de Matriz
{ }	Definição de Matriz, Constante ou Bloco de Código
->	Identificador de Apelido
&	Macrosubstituição
@	Passagem de parâmetro por referência

Os parênteses são utilizados para agrupar elementos em uma expressão mudando a ordem de precedência da avaliação da expressão (segundo as regras matemáticas por exemplo). Também servem para envolver os argumentos de uma função. Veja a documentação sobre [precedência de operadores](#) para maiores detalhes.

Os colchetes são utilizados para especificar um elemento específico de uma matriz. Por exemplo, A[3,2], refere-se ao elemento da matriz A na linha 3, coluna 2.

As chaves são utilizadas para a especificação de matrizes literais ou blocos de código. Por exemplo, A:={10,20,30} cria uma matriz chamada A com três elementos.

O símbolo -> identifica um campo de um arquivo diferenciando-o de uma variável. Por exemplo, FUNC->nome refere-se ao campo nome do arquivo FUNC. Mesmo que exista uma variável chamada nome, é o campo nome que será acessado.

O símbolo & identifica uma avaliação de expressão através de macro e é visto em detalhes na documentação sobre [macrosubstituição](#).

O símbolo @ é utilizado para indicar que durante a passagem de uma variável para uma função ou procedimento ela seja tomada como uma referência e não como valor.

Ordem de Precedência dos Operadores

Dependendo do tipo de operador, existe uma ordem de precedência para a avaliação dos operandos. Em princípio, todas as operações com os operadores são realizadas da esquerda para a direita se eles tiverem o mesmo nível de prioridade.

A ordem de precedência, ou nível de prioridade de execução, dos operadores em AdvPI é:

1. Operadores de Incremento/Decremento pré-fixado
2. Operadores de String
3. Operadores Matemáticos
4. Operadores Relacionais
5. Operadores Lógicos
6. Operadores de Atribuição
7. Operadores de Incremento/Decremento pós-fixado

Em expressões complexas com diferentes tipos de operadores, a avaliação seguirá essa sequência. Caso exista mais de um operador do mesmo tipo (ou seja, de mesmo nível), a avaliação se dá da esquerda para direita. Para os operadores matemáticos entretanto há uma precedência a seguir:

1. Exponenciação
2. Multiplicação e Divisão
3. Adição e Subtração

Considere o exemplo:

Local nResultado := 2+10/2+5*3+2^3

O resultado desta expressão é 30, pois primeiramente é calculada a exponenciação $2^3 (=8)$, então são calculadas as multiplicações e divisões $10/2 (=5)$ e $5*3 (=15)$, e finalmente as adições resultando em $2+5+15+8 (=30)$.

Alteração da Precedência

A utilização de parênteses dentro de uma expressão altera a ordem de precedência dos operadores. Operandos entre parênteses são analisados antes dos que se encontram fora dos parênteses. Se existirem mais de um conjunto de parênteses não-aninhados, o grupo mais a esquerda será avaliado primeiro e assim sucessivamente.

Local nResultado := (2+10)/(2+5)*3+2^3

No exemplo acima primeiro será calculada a exponenciação $2^3 (=8)$. Em seguida $2+10 (=12)$ será calculado, $2+5 (=7)$ calculado, e finalmente a divisão e a multiplicação serão efetuadas, o que resulta em $12/7 * 3 + 8 (=13.14)$.

Se existirem vários parênteses aninhados, ou seja, colocados um dentro do outro, a avaliação ocorrerá do parênteses mais interno em direção ao mais externo.

Macro Substituição

O operador de macro substituição, simbolizado pelo "e" comercial (&), é utilizado para a avaliação de expressões em tempo de execução. Funciona como se uma expressão armazenada fosse compilada em tempo de execução, antes de ser de fato executada. Considere o exemplo:

```
01 X := 10
```

```
02 Y := "X + 1"
```

```
03 B := &Y // O conteúdo de B será 11
```

A variável X é atribuída com o valor 10, enquanto a variável Y é atribuída com a string de caracteres contendo "X + 1".

A terceira linha utiliza o operador de macro. Esta linha faz com que o número 11 seja atribuído à variável B. Pode-se perceber que esse é o valor resultante da expressão em formato de caractere contida na variável Y.

Utilizando-se uma técnica matemática elementar, a substituição, temos que na segunda linha, Y é definido como "X + 1", então pode-se substituir Y na terceira linha:

```
03 B := &"X + 1"
```

O operador de macro cancela as aspas:

```
03 B := X + 1
```

Pode-se perceber que o operador de macro remove as aspas, o que deixa um pedaço de código para ser executado. Deve-se ter em mente que tudo isso acontece em tempo de execução, o que torna tudo muito dinâmico. Uma utilização interessante é criar um tipo de calculadora, ou avaliador de fórmulas, que determina o resultado de algo que o usuário digita.

-

O operador de macro tem uma limitação: variáveis referenciadas dentro da string de caracteres (X nos exemplos anteriores) não podem ser locais.

Estruturas de Controle

Controlando o Fluxo

O AdvPI suporta várias estruturas de controle que permitem mudar a sequência de fluxo de execução de um programa. Estas estruturas permitem a execução de código baseado em condições lógicas e a repetição da execução de pedaços de código qualquer número de vezes.

Em AdvPI, todas as estruturas de controle podem ser "aninhadas" dentro de todas as demais estruturas contanto que estejam aninhadas propriamente. Estruturas de controle têm um identificador de início e um de fim, e qualquer estrutura aninhada deve se encontrar entre estes identificadores.

Também existem estruturas de controle para determinar que elementos, comandos, etc em um programa serão compilados. Estas são as diretivas do pré-processador `#ifdef...#endif` e `#ifndef...#endif`. Consulte a documentação sobre o pré-processador para maiores detalhes.

As estruturas de controle em AdvPI estão divididas em [Estruturas de Repetição](#) e [Estruturas de Decisão](#).

Estruturas de Repetição

Repetição de Comandos

Estruturas de repetição são designadas para executar uma seção de código mais de uma vez. Por exemplo, imaginando-se a existência de uma função para imprimir um relatório, pode-se desejar imprimi-lo quatro vezes. Claro, pode-se simplesmente chamar a função de impressão quatro vezes em sequência, mas isto se tornaria pouco profissional e não resolveria o problema se o número de relatórios fosse variável.

Em AdvPI existem dois comandos para a repetição de seções de código. O comando [FOR...NEXT](#) e o comando [WHILE...ENDDO](#).

O Comando FOR...NEXT

A estrutura de controle FOR...NEXT, ou simplesmente o loop FOR, repete uma seção de código em um número determinado de vezes.

Sintaxe

FOR *Variavel* := *nValorInicial* TO *nValorFinal* [STEP *nIncremento*]

Comandos...

[EXIT]

[LOOP]

NEXT

Parâmetros

Variavel	Especifica uma variável ou um elemento de uma matriz para atuar como um contador. A variável ou o elemento da matriz não precisa ter sido declarado antes da execução do comando FOR...NEXT. Se a variável não existir, será criada como uma variável privada .
nValorInicial TO nValorFinal	nValorInicial é o valor inicial para o contador; nValorFinal é o valor final para o contador. Pode-se utilizar valores numéricos literais, variáveis ou expressões, contanto que o resultado seja do tipo de dado numérico.
STEP nIncremento	nIncremento é a quantidade que será incrementada ou decrementada no contador após cada execução da seção de comandos. Se o valor de nIncremento for negativo, o contador será decrementado. Se a cláusula STEP for omitida, o contador será incrementado em 1. Pode-se utilizar valores numéricos literais, variáveis ou expressões, contanto que o resultado seja do tipo de dado numérico.
Comandos	Especifica um ou mais instruções de comando AdvPI que serão executadas.
EXIT	Transfere o controle de dentro do comando FOR...NEXT para o comando imediatamente seguinte ao NEXT, ou seja, finaliza a repetição da seção de comandos imediatamente. Pode-se colocar o comando EXIT em qualquer lugar entre o FOR e o NEXT.
LOOP	Retorna o controle diretamente para a cláusula FOR sem executar o restante dos comandos entre o LOOP e o NEXT. O contador é incrementado ou decrementado normalmente, como se o NEXT tivesse sido alcançado. Pode-se colocar o comando LOOP em qualquer lugar entre o FOR e o NEXT.

Comentários

Uma variável ou um elemento de uma matriz é utilizado como um contador para especificar quantas vezes os comandos AdvPI dentro da estrutura FOR...NEXT são executados. Os comandos AdvPI depois do FOR são executados até que o NEXT seja alcançado. O contador (Variavel) é então incrementado ou decrementado com o valor em nIncremento (se a cláusula STEP for omitida, o contador é incrementado em 1). Então, o contador é comparado com o valor em nValorFinal. Se for menor ou igual ao valor em nValorFinal, os comandos seguintes ao FOR são executados novamente. Se o valor for maior que o contido em nValorFinal, a estrutura FOR...NEXT é terminada e o programa continua a execução no primeiro comando após o NEXT.

Os valores de nValorInicial, nValorFinal e nIncremento são apenas considerados inicialmente. Entretanto, mudar o valor da variável utilizada como contador dentro da estrutura afetará o número de vezes que a repetição será executada. Se o valor de nIncremento é negativo e o valor de nValorInicial é maior que o de nValorFinal, o contador será decrementado a cada repetição.

Exemplo

Local nCnt

Local nSomaPar := 0

```
For nCnt : = 0 To 100 Step 2
    nSomaPar += nCnt
Next

Alert( "A soma dos 100 primeiros números pares é: " + ;
    cValToChar(nSomaPar) )

Return
```

Este exemplo imprime a soma dos 100 primeiros números pares. A soma é obtida através da repetição do cálculo utilizando a própria variável de contador. Como a cláusula STEP está sendo utilizada, a variável nCnt será sempre incrementada em 2. E como o contador começa com 0, seu valor sempre será um número par.

O Comando WHILE...ENDDO

A estrutura de controle WHILE...ENDDO, ou simplesmente o loop WHILE, repete uma seção de código enquanto uma determinada expressão resultar em verdadeiro (.T.).

Sintaxe

```
WHILE IExpressao
    Comandos...
[EXIT]
[LOOP]
ENDDO
```

Parâmetros

IExpressao	Especifica uma expressão lógica cujo valor determina quando os comandos entre o WHILE e o ENDDO são executados. Enquanto o resultado de IExpressao for avaliado como verdadeiro (.T.), o conjunto de comandos são executados.
Comandos	Especifica um ou mais instruções de comando AdvPI que serão executadas enquanto IExpressao for avaliado como verdadeiro (.T.).
EXIT	Transfere o controle de dentro do comando WHILE...ENDDO para o comando imediatamente seguinte ao ENDDO, ou seja, finaliza a repetição da seção de comandos imediatamente. Pode-se colocar o comando EXIT em qualquer lugar entre o WHILE e o ENDO.

LOOP

Retorna o controle diretamente para a cláusula WHILE sem executar o restante dos comandos entre o LOOP e o ENDDO. A expressão em IExpressao é reavaliada para a decisão se os comandos continuarão sendo executados.

Comentários

Os comandos entre o WHILE e o ENDDO são executados enquanto o resultado da avaliação da expressão em IExpressao permanecer verdadeiro (.T.). Cada palavra chave WHILE deve ter uma palavra chave ENDDO correspondente.

Exemplo

```
Local nNumber := nAux := 350
```

```
nAux := Int(nAux / 2)
```

```
While nAux > 0
```

```
    nSomaPar += nCnt
```

```
Next
```

```
Alert( "A soma dos 100 primeiros números pares é: " + ;
```

```
    cValToChar(nSomaPar) )
```

```
Return
```

Estruturas de Decisão

Desviando a Execução

Estruturas de desvio são designadas para executar uma seção de código se determinada condição lógica resultar em verdadeiro (.T.). Em AdvPI existem dois comandos para execução de seções de código de acordo com avaliações lógicas. O comando [IF...ENDIF](#) e o comando DO CASE...ENDCASE.

O Comando IF...ENDIF

Executa um conjunto de comandos baseado no valor de uma expressão lógica.

Sintaxe

```
IF I Expressao
    Comandos
[ELSE
    Comandos. . . ]
ENDIF
```

Parâmetros

IExpressao	<p>Especifica uma expressão lógica que é avaliada. Se IExpressao resultar em verdadeiro (.T.), qualquer comando seguinte ao IF e antecedente ao ELSE ou ENDIF (o que ocorrer primeiro) será executado.</p> <p>Se IExpressao resultar em falso (.F.) e a cláusula ELSE for definida, qualquer comando após essa cláusula e anterior ao ENDIF será executada. Se a cláusula ELSE não for definida, todos os comandos entre o IF e o ENDIF são ignorados. Neste caso, a execução do programa continua com o primeiro comando seguinte ao ENDIF.</p>
Comandos	Conjunto de comandos AdvPI que serão executados dependendo da avaliação da expressão lógica em IExpressao.

Comentários

Pode-se aninhar um bloco de comando IF...ENDIF dentro de outro bloco de comando IF...ENDIF. Porém, para a avaliação de mais de uma expressão lógica, deve-se utilizar o comando [DO CASE...ENDCASE](#).

Exemplo

```
Local dVencTo := CTOD("31/12/01")
If Date() > dVencTo
    Alert("Vencimento ultrapassado!")
Endif
Return
```

O Comando DO CASE...ENDCASE

Executa o primeiro conjunto de comandos cuja expressão condicional resulta em verdadeiro (.T.).

Sintaxe

```
DO CASE
    CASE I Expressao1
        Comandos
    [CASE I Expressao2
        Comandos
    ...
    CASE I ExpressaoN
        Comandos]
    [OTHERWISE
        Comandos]
ENDCASE
```

Parâmetros

CASE IExpressao1 Comandos...	<p>Quando a primeira expressão CASE resultante em verdadeiro (.T.) for encontrada, o conjunto de comandos seguinte é executado. A execução do conjunto de comandos continua até que a próxima cláusula CASE, OTHERWISE ou ENDCASE seja encontrada. Ao terminar de executar esse conjunto de comandos, a execução continua com o primeiro comando seguinte ao ENDCASE.</p> <p>Se uma expressão CASE resultar em falso (.F.), o conjunto de comandos seguinte a esta até a próxima cláusula é ignorado.</p> <p>Apenas um conjunto de comandos é executado. Estes são os primeiros comandos cuja expressão CASE é avaliada como verdadeiro (.T.). Após a execução, qualquer outra expressão CASE posterior é ignorada (mesmo que sua avaliação resultasse em verdadeiro).</p>
OTHERWISE Commandos	<p>Se todas as expressões CASE forem avaliadas como falso (.F.), a cláusula OTHERWISE determina se um conjunto adicional de comandos deve ser executado. Se essa cláusula for incluída, os comandos seguintes serão executados e então o programa continuará com o primeiro comando seguinte ao ENDCASE. Se a cláusula OTHERWISE for omitida, a execução continuará normalmente após a cláusula ENDCASE.</p>

Comentários

O Comando DO CASE...ENDCASE é utilizado no lugar do comando [IF...ENDIF](#) quando um número maior do que uma expressão deve ser avaliada, substituindo a necessidade de mais de um comando IF...ENDIF aninhados.

Exemplo

```

Local nMes      := Month(Date())
Local cPeriodo := ""

DO CASE
  CASE nMes <= 3
    cPeriodo := "Primeiro Trimestre"
  CASE nMes >= 4 .And. nMes <= 6
    cPeriodo := "Segundo Trimestre"
  CASE nMes >= 7 .And. nMes <= 9
    cPeriodo := "Terceiro Trimestre"
  OTHERWISE
    cPeriodo := "Quarto Trimestre"
ENDCASE

Return

```

Informações Adicionais

Lista de 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	
DTOD	IIF	RECNO	UPPER	

Notas:

- Palavras reservadas não podem ser utilizadas para variáveis, procedimentos, ou funções.
- Funções reservadas são pertencentes ao compilador e portanto não podem ser redefinidas por uma aplicação.
- Abreviações de quatro letras de palavras reservadas e funções também são reservadas.
- Todos os identificadores que começarem com um ou mais caracteres de sublinhado (_) são utilizados como identificadores internos e portanto são também reservados.

Técnicas de Programação Eficiente em AdvPI

Para o desenvolvimento de sistemas e a programação de rotinas, sempre é esperado que qualquer código escrito seja:

- de correto funcionamento

- eficiente
- legível
- reutilizável
- extensível
- portátil

Após anos de experiência na utilização de linguagens padrão xBase e do desenvolvimento da linguagem AdvPI, algumas técnicas para uma programação otimizada e eficiente foram reconhecidas. A utilização das técnicas a seguir, visa buscar o máximo aproveitamento dos recursos da linguagem com o objetivo de criar programas com estas características.

Criação de Funções Segundo a Necessidade

Observe o código de exemplo:

```
User Function GetAnswer(IDefault)
```

```
Local IOk
```

```
IOk := GetOk(IDefault)
```

```
If IOk
```

```
Return .T.
```

```
Else
```

```
Return .F.
```

```
Endif
```

```
Return nil
```

Utilizando-se apenas o critério "a função funciona corretamente?", a função GetAnswer é perfeita. Recebe um parâmetro lógico com a resposta padrão e retorna um valor lógico dependente da opção escolhida pelo usuário em uma função de diálogo "sim/não" designada para isso. Pode entretanto ser melhorada, particularmente se eficiência for considerada como um critério para um código melhor. Eficiência tipicamente envolve a utilização de poucos recursos de máquina, poucas chamadas de funções ou tornar mais rápido um processo.

Segundo esse raciocínio, poderia se produzir o seguinte código:

```
User Function GetAnswer(IDefault)
```

```
Return If( GetOk(IDefault), .T., .F.)
```

Ou melhor:

```
User Function GetAnswer(IDefault)
```

```
Return GetOk(IDefault)
```

Com a otimização do código da função GetAnswer, pode facilmente verificar que a mesma não realiza nada adicional à chamada de GetOk, podendo ser substituída por uma chamada direta desta, continuando a funcionar corretamente.

Codificação Auto-Documentável

Nenhum comentário substitui um código claramente escrito, e este não é um acidente. Considere o exemplo:

```
cVar := " " // 11 espaços
```

O tamanho da variável cVar não é evidente por si só e não é facilmente verificado. Estes mesmos 10 espaços estariam mais óbvios e ainda assim garantidos se a instrução fosse escrita como:

```
cVar := Space(10)
```

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 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 "!!!!!!!!!!"
```

Que pode ficar mais simples assim:

```
@ 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 Identação

Este é um hábito que todo programador deve desenvolver. Não consome muito esforço para manter o código alinhado durante o trabalho, porém quando necessário pode-se utilizar AP6 IDE para a reidentação de código.

Considere o exemplo:

```
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, case, 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
```

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. 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
```

Utilização da Notação Húngara

A Notação Húngara é muito comum entre programadores xBase e de outras linguagens. A documentação do AdvPI 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 Notação Húngara disponível na documentação da linguagem AdvPI.

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. 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 AdvPI 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 (B1_FILIAL+B1_COD+B1_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 AdvPI 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 frequentemente 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 RaizQuadrada( 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.

Tabela de Pictures de Formatação

Comando SAY/PSAY

Funções	
C	Exibe CR depois de números positivos
E	Exibe numéricos com o ponto e a vírgula invertidos (formato Europeu)
R	Inserir caracteres diferentes dos caracteres de template
X	Exibe DB depois de números negativos
Z	Exibe zeros como brancos
(Envolve números negativos entre parênteses
!	Converte todos os caracteres alfabéticos para maiúsculo

Templates	
X	Exibe dígitos para qualquer tipo de dado
9	Exibe dígitos para qualquer tipo de dado
#	Exibe dígitos para qualquer tipo de dado
!	Converte caracteres alfabéticos para maiúsculo
*	Exibe asterisco no lugar de espaços em branco iniciais em números
.	Exibe a posição do ponto decimal
,	Exibe a posição do milhar

Comando GET

Funções	
A	Permite apenas caracteres alfabéticos
C	Exibe CR depois de números positivos
E	Exibe numéricos com o ponto e vírgula invertidos (formato Europeu)
R	Insere caracteres diferentes dos caracteres de template na exibição mas não insere-os 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	
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

Programação do AdvPI para o ERP Siga Advanced Protheus

Prefácio

Existe um ditado chinês que diz: “O Homem não tropeça em montanhas, tropeça em pedregulhos, areia, pequenos buracos, mas nunca em uma montanha”.

Isso nos remete a pensar que onde erramos é exatamente no simples, naquele detalhe quase imperceptível e que tem um valor muito grande para o todo. Avaliemos do ponto de vista humano: será tão difícil cumprimentar a todos, sermos mais amigos, mais serenos nas decisões e companheiros uns dos outros e trabalharmos em equipe? Por que muitas vezes não o fazemos? Por que insistimos no individualismo e no mal-humor? Não seria mais fácil, até mesmo óbvio, estarmos mais bem-humorados e dispostos a trabalhar em equipe, trocarmos conhecimento e discernimento nas decisões, pensarmos mais no todo porém se importando com as partes que o compõe?

Seria mais interessante se ao caminharmos por um parque, prestássemos mais atenção nas árvores, no caminho, nas flores, no canto dos passarinhos sem se esquecer do objetivo do passeio, sem perder a noção de tempo e distância, mas curtindo muito a paisagem, o detalhe.

Agora vamos traçar um paralelo com o nosso dia a dia. Não seria melhor ao reservarmos um fonte, verificarmos com mais atenção:

As condicionais? Afinal muitas vezes não testamos um ELSE.

Os filtros? Geralmente esquecemos de tentar otimizar a performance no SQL.

As mensagens? Afinal é tão comum nos depararmos com textos completamente sem sentido.

Os helps? Damos pouca atenção a eles e nos esquecemos que é a primeira coisa que o usuário tenta.

Imaginem algumas ligações menos por causa de uma simples documentação a mais! Aquele ponto de entrada que criamos e não pensamos nos supostos parâmetros que nosso pessoal em campo pode querer, ou mesmo no retorno mais adequado para aquela função.

Lembrem-se também da documentação do novo campo; Ela realmente é necessária? Se a chave de índice é imprescindível, por que não crio uma query? Ao responder um BOPS, não seria melhor que fosse sua última argumentação para o problema? Se isto ficar claro e bem resolvido não teremos mais aquela ocorrência ou dúvida. Se tivermos que explicar um processo para alguém, que o façamos de tal forma a não gerarmos incógnitas.

Por que ao invés de focarmos nossos esforços para “matarmos” o BOPS, não avaliamos o fonte para evitarmos NOVOS BOPS? Ao resolver uma ocorrência lembre-se de todos os pontos de implicação da sua atividade. O que isso irá impactar no serviço do outro? Sem falar em documentar no Quark!

Vamos trazer o comportamento do parque para o nosso trabalho também. Ao programar vamos nos ater aos detalhes, sermos mais críticos, pensarmos que aquela instrução a mais, significa muito para o sistema e que lá na frente, se tratado com descuido, pode causar problemas.

Tenha convicção que, se agirmos de maneira mais focada aos nossos propósitos, o passeio ou melhor a programação, será muito mais entusiasmada, produtiva e com uma margem de erro bem menor. Com esse comportamento quem ganha somos nós; Microsigla!. Só assim teremos mais tempo de irmos ao parque no final de semana.

Lembre-se que não adianta decidirmos passear no parque do Ibirapuera no domingo, e não estarmos com a cabeça voltada para o passeio, ao invés disso pensarmos no trabalho, na DLLI que não comunica, no BOPS que não foi baixado, pois se assim for, estaremos tão voltados para outros fins que não curtiremos o passeio. Pense que para passear, ou melhor, programar, a regra também é válida, não adianta nem ao menos tentarmos se não estivermos concentrados para isso.

Enfim, quer uma prova de trabalho em equipe com um alto nível de qualidade e detalhes: este manual, que foi constituído em apenas 2 dias, com a colaboração de mais de 20 pessoas, focadas em seus objetivos, se atentando cada um com o seu tema. O resultado? Um trabalho excelente, um documento para nos ajudar a sermos melhores e não errarmos no fácil!

O Que é Fazer um Programa com Inteligência

Precisamos entender, antes de mais nada, o que é inteligência.

Segundo o dicionário Michaelis, inteligência significa:

faculdade de entender, pensar, raciocinar e interpretar;

Compreensão, conhecimento profundo.

De acordo com essa definição, se pretendemos utilizar nosso bem mais precioso em nosso trabalho, vamos precisar desenvolver alguns hábitos:

Devemos estudar o programa antes de começar a desenvolver. Imagine prestar um concurso ou fazer uma prova sem estudar. Vai ganhar um zero na certa! No programa não será diferente!

Fazer um levantamento dos programas que sofrerão as consequências das alterações realizadas. Todos esses programas deverão ser testados juntamente com o programa alterado.

Antes de criar uma função, consulte o Help Microsiga ou os colegas de trabalho, pois esta função já pode ter sido criada.

Ao criar uma função, certifique-se de que no cabeçalho conste algumas informações básicas como: descrição da função, sintaxe, definição dos parâmetros e autor. É comum ao desenvolver uma função, utilizarmos outra já pronta como exemplo, e neste momento o "copiar/colar" nos faz esquecer de alterar estas informações.

Imagine se alguém desenvolver uma função inconsistente e esquecer de trocar o seu nome no cabeçalho. Devemos assumir a responsabilidade de nossos atos.

Ao fazer a documentação das alterações realizadas, certifique-se de que as informações estão claras, não só para o seu entendimento mas para que os colegas não percam tempo tentando entender-las.

Ao realizar os testes, defina critérios. Antes de começar defina onde quer chegar. Não basta consistir suas alterações. O fato de suas alterações estarem funcionando como previstas não garante a não existência de erros.

Não limite-se a testar sua alteração na base que você utilizou durante o desenvolvimento, pois você criou o ambiente perfeito para que o programa funcione.

Pode parecer um pouco trabalhoso passar por estes processos no decorrer do desenvolvimento do sistema, mas se medidas como estas não forem tomadas, o que era extremamente simples se tornará extremamente trabalhoso.

Programando Simples, mas Certo

Qual profissional da área de informática ainda não se deparou com um código fonte que parecia estar escrito em outro dialeto mesmo com todo conhecimento adquirido naquela linguagem, este fato geralmente ocorre pela má utilização de sintaxes complexas que nem sempre significam um bom funcionamento do sistema.

Um profissional da área de informática não possui nenhum modelo padrão para desenvolver os seus algoritmos, porém é necessária a aplicação da ética profissional para que se possa desenvolver algoritmos de maneira simples e correta, este conceito se baseia nos seguintes aspectos :

Entender qual o objetivo do processo em questão

Analisar a melhor forma de desenvolver um algoritmo que seja de fácil manutenção.

Utilizar comandos e sintaxes que utilizem o máximo de simplicidade e clareza possível.

Erros que Podem ser Evitados

Existem alguns erros que com um pouco de atenção, podem ser evitados, tais como:

Verifique se a variável está declarada antes do uso;

Ao declarar uma variável, verifique qual a necessidade de ter essa variável e qual o tipo e a sua classe;

Classifiquem as funções e os procedimentos conforme a necessidade, como por exemplo, na declaração de um array, defina o seu tamanho e no uso verifique se o elemento existe;

Salve a ordem e a área e o registro do arquivo que será utilizado para que no final do processo se recupere estes valores;

Evite retornar da função antes do seu final, ou seja, crie preferencialmente um único retorno;

Valide sempre o retorno do ponto de entrada;

Quando for gravar um arquivo que utiliza campos de outros arquivos, posicione todos os arquivos e registros antes de iniciar a gravação, e descreva o alias do campo;

Utilize de arquivo CH nas strings para localização;

Quando possível utilize a linguagem SQL, pois minimiza o tempo de execução em muitos processos.

A Importância de Programas Documentados

Todos sabemos o quanto é difícil elaborar e manter uma documentação técnica atualizada, ainda mais aqui na Microsiga, cuja dinâmica dos acontecimentos muitas vezes impede que isso seja viabilizado. Diante desse cenário, o que nos resta? Obviamente que pelo menos os programas sejam documentados, bem documentados.

Documentar bem, não significa que tenhamos que escrever dezenas de linhas de comentários a cada linha de código. Significa que os comentários têm passar alguma informação relevante. Vemos comentários assim: "compara A com B" e só. Isso é óbvio, a leitura do código já nos diz isso. A documentação deve se ater a conceitos, por exemplo: "Se A for maior que B, o arquivo de saldos será atualizado, caso contrário o registro será rejeitado para que o saldo não fique negativo.". Isto sim transmite alguma informação.

Também se pode utilizar desse recurso para fazer lembretes a fatos importantes que, se forem deixados de lado, podem comprometer o funcionamento das rotinas.

Por exemplo: "Ao acionar esta função, o arquivo XXX DEVE estar posicionado no índice 1".

E os cabeçalhos? Quantos programas são "aproveitados" e nem sequer o nome do autor é trocado? Se o analista X tivesse escrito todos programas que aparece como autor ele deveria ter começado na época do Charles Babage. O cabeçalho das funções de conter o nome na dita cuja, autor, data de criação, uma descrição sumária de sua funcionalidade, a sintaxe e por último, mas não menos importante, a descrição dos argumentos de entrada e saída. A respeito desse último item deve-se ter especial atenção nas manutenções, pois novos argumentos são criados e nem sempre são declarados nessa seção da documentação do cabeçalho, isso é muito grave.

No IDE do PROTHEUS existem opções bastante interessantes para nos auxiliar nessa tarefa. Experimente as opções Inserir, Documentação de cabeçalho e Inserir, Documentação de Explicação.

Existe ainda um tipo de documentação que nem sempre é observada, é aquela inerente ao próprio código. Programas cujas variáveis são declaradas como nX, cVAR1, dAUX, nNUM, etc., são extremamente difíceis de entender e pior, manter. É conveniente que os nomes das variáveis retratem seu uso ou destino. Por exemplo: dDataDeS ou dDataDeE. Segundo as convenções da Microsiga, variáveis do tipo DATA devem ser iniciadas pela letra "d". Assim "Data", não acrescenta nada ao entendimento do que a variável representa. Nos sobrou o "dES" e o "dEE" para informar para que diados serve a bendita variável. Será saída, solução, saldo? Entrada, Estorno, Estoque? Que tal isso: dSeguro e dEntrega?

Enfim, como foi dito, não é preciso escrever um livro a cada programa, basta ser objetivo e se colocar na posição de quem não conhece o programa tão pouco o assunto. Algum dia você mesmo poderá estar nessa posição.

Cabeçalho de Programa / Função

O cabeçalho do programa é utilizado para identificar informações gerais sobre a rotina, seu autor, data, entre outras informações. É importante que esteja preenchida de forma correta e atualizada. Lembre-se de que nada adianta um cabeçalho que não informe nada ou pior ainda, com informações errôneas.

Lembre-se que um bom livro começa com um bom prefácio, e um bom programa começa com um cabeçalho útil e legível.

A manutenção/atualização do cabeçalho é de responsabilidade da última pessoa que alterou o fonte. O cabeçalho de programa padrão da Microsiga contém: rotina, autor, data do desenvolvimento, comentário sintético e sintaxe.

Criação de Variáveis

Na criação de uma variável deve-se ter em mente alguns pontos fundamentais:

- A declaração
- O tipo de variável
- A função CRIAVAR()
- A inicialização
- Padronização de variáveis

A Declaração

Deve ser feita sempre no início da rotina que for utilizá-la, como no exemplo:

```
Function a910VerCod()  
  
Local cCod910 := "001"  
  
Return
```

O Tipo de Variável

O tipo de variável serve para identificar a utilização que a mesma terá no decorrer da rotina. Toda variável deve estar tipada durante sua criação. Quando programamos nativamente em "C", isto se torna obrigatório. Devemos fazer o mesmo no AP5, pois isto demonstra que a variável foi conscientemente declarada.

Tipos Existentes

PUBLIC: Esta variável será inicializada em um valor lógico falso (.F.) até que seja atribuído um valor específico a ela. Esta variável permanece definida por toda a duração da aplicação e pode ser vista (assim como usada, alterada e avaliada) por qualquer função. Esta variável gera um token (indicação) na tabela de símbolos, isto significa que o módulo principal conterà símbolos para esta classe de variável, o que, por sua vez, ocupa mais espaço de memória. Deve-se evitar a utilização deste tipo, a não ser em casos extremos.

PRIVATE: Esta variável será inicializada em valor nulo (NIL) e uma vez declarada, permanecerá assim durante toda a duração do fluxo da função, até que este volte ao procedimento inicial que a chamou. Em essência, uma variável de memória PRIVATE inicializada logo no início do Protheus, agirá como um variável PUBLIC. Esta variável pode ser vista por uma sub-rotina da função e modificada de maneira correspondente. Esta variável também gera um token na tabela de símbolos comentada acima.

LOCAL: Esta variável de memória será inicializada com valor nulo (NIL) e só é visível dentro da função que a inicializa, mesmo que esta última, contenha funções incorporadas a seu conteúdo. Este tipo de variável é o mais adequado a ser utilizado em funções, pois não gera símbolos na tabela de símbolos, por consequência ocupa pouco espaço de memória e, o compilador avalia as variáveis LOCAL e STATIC mais

rapidamente que os outros tipos (PUBLIC e PRIVATE). Cuidado para não sucumbir à teoria de que se pode obter economia de memória, mudando qualquer referência PRIVATE para uma referência LOCAL. Se você fizer isso, as funções podem não funcionar corretamente, embora funcionassem na versão anterior às alterações.

STATIC: A variável STATIC é idêntica à classe de armazenamento LOCAL, com uma exceção. Uma variável STATIC é retida dentro de sua sub-rotina, mesmo depois que o fluxo da função a tenha deixado. Isto é particularmente útil para funções independentes tipo "caixa-preta", que contêm seu próprio conjunto de variáveis exclusivas e devem manter esses valores de interação em interação.

Inicialização

Quando não atribuímos nenhum valor a uma variável no momento de sua declaração, corremos o risco de utilizá-la com valor "NIL" e causar erros fatais. Por isso, a inicialização de uma variável é de extrema importância.

Padronização de Variáveis

É importante que ao lermos o nome de uma variável, possamos saber se o seu tipo é numérico, caracter, data ou lógico. O nome da variável de get não deve coincidir com uma variável de outro programa, pois toda variável de get possui um help específico.

Exemplo:

a variável DBaixa (get da baixa no programa de Títulos a Receber), já possui um texto help que indica seu conteúdo e não deverá ser criada outra variável para outra finalidade com este mesmo nome.

Para tanto, definimos a seguinte padronização :

N -> Numéricas

L -> Lógicas

D -> Data

C -> Caracter

A -> Array (matriz)

O -> Objeto

U -> Sem definição

Criando uma Variável Utilizando a Função CRIAVAR()

Esta função cria uma variável, retornando o valor do campo, de acordo com o dicionário de dados. Avalia o inicializador padrão e retorna o conteúdo de acordo com o tipo de dado definido no dicionário.

Sintaxe

uRet : = CriaVar(cCampo, lIniPad, cLado)

Onde :

Uret -> tipo de retorno de acordo com o dicionário de dados, considerando inicializador padrão.

cCampo -> Nome do campo

IniPad -> Indica se considera (.T.) ou não (.F.) o inicializador padrao (X3_RELACAO)

Clado -> Lado para inicialização padrão

Variáveis de Relatórios

Na criação de um relatório algumas variáveis e seus tipos são convencionados para a utilização da biblioteca de funções de relatório.

Variável	Tipo	Conteúdo
wnRel	Local	Nome default do relatório em disco
cbCont	Local	Contador
Cabec1	Local	1ª linha do cabeçalho do relatório
Cabec2	Local	2ª linha do cabeçalho do relatório
Cabec3	Local	3ª linha do cabeçalho do relatório
Tamanho	Local	Tamanho do Relatório (P = Pequeno 80 colunas, M = Médio 132 colunas, G = Grande, 220 colunas)
cDesc1	Local	1ª linha da descrição do relatório
cDesc2	Local	2ª linha da descrição do relatório
cDesc3	Local	3ª linha da descrição do relatório
Limite	Local	Quantidade de colunas no relatório (80,132,220)
Titulo	Local	Título do Relatório
aReturn	Private	Matriz com as informações para a tela de configuração de impressão
Nomeprog	Private	Nome do programa do relatório
cString	Private	Alias do arquivo principal do relatório para o uso de filtro
Li	Private	Controle das linhas de impressão. Seu valor inicial é a quantidade máxima de linhas por página utilizada no relatório
m_pag	Private	Controle do número de páginas do relatório
aOrd	Private	Matriz contendo as ordens de layout para a impressão. Caso não existam várias ordens esta matriz deve estar vazia. Ex.: aOrd := {"Código", "Descrição", "Telefone"} -> O layout do relatório vai depender da ordem selecionada na tela de configuração de impressão
nLastKey	Private	Utilizado para controlar o cancelamento da impressão do relatório
cPerg	Private	Nome da pergunta a ser exibida para o usuário
aLinha	Private	Matriz que contém informações para impressão de relatórios cadastrais

Cuidados com o Posicionamento de Registros

O posicionamento correto de registros é fundamental para a funcionalidade completa dos programas. Algumas dicas para posicionamento de registros são :

Evitar `DBGOTOP()`, usar `DBSEEK(XFILIAL())` para os arquivos de dados do sistema. O comando `DBGOTOP()` somente será utilizado quando da real necessidade de se efetuar uma leitura desde o início do arquivo independente do tratamento de filial.

Como no mesmo arquivo de dados, poderemos ter registros de várias filiais, desta forma ficará garantido o posicionamento no primeiro registro da filial corrente.

Ao executar um `DBSEEK()`, verificar se localizou o registro, exemplo:

```
If ! SB1->(dbSeek(xFilial("SB1")))
    // Não achei o registro
```

```
Endif
```

Mesmo que seja óbvio a existência do registro, faça o teste pois o programa deve prever que a base de dados não é tão confiável como deveria, e um alerta ajuda a identificar estes casos. Em casos de relatórios, atentar-se para imprimir a mensagem de forma consciente.

Se for executada a função `RECLOCK(cAlias, .F.)`, para alteração do registro atual, em um arquivo no estado de `EOF()` (caso falhe um `DBSEEK()`) será abortado o programa e gravado um arquivo texto de nome `MSRLOCK.EOF` que poderá ser usado para averiguações.

O comando `SOFTSEEK` determina se será usada uma busca relativa durante uma procura em um banco de dados. Se este comando estiver em `ON`, e for utilizada uma função `DBSEEK()`, e nenhuma correspondência for encontrada, o ponteiro de registro ficará no próximo registro do índice que possua um valor mais alto que a expressão utilizada nesta função. Este comando deverá ser utilizado com a máxima atenção, pois caso esteja ligado, poderá localizar um registro errado.

Quanto ao comando `DO WHILE` não esquecer de incluir a condição referente à filial, quando esta leitura for de registros de uma filial). Exemplo :

```
dbSelectArea("SB1")
dbSeek(xFilial("SB1"))
Do While ! Eof() .And. B1_FILIAL == xFilial("SB1")
    // Processamento
    dbSkip()
Enddo
```

Ao criar uma função que irá desposicionar registros, use a função `GETAREA()` e `RESTAREA()`, para voltar tudo à posição original. Exemplo:

```
Dbselectarea("SD1")
aAreasd1 := Getarea() // Armazena o ambiente do arquivo SD1
SD1->(dbsetorder(3))
SD1->(dbseek(xfilial("SD1") + DTOS("01/03/01"), .T.))
Do While ! Eof() .And. D1_FILIAL == xfilial("SD1") .And. DTOS(D1_EMISSAO) <= DTOS(mv_par02)
```

```
// Processamento  
  
Dbskip()  
  
Enddo  
  
Restarea(aAreasd1) // Restaura o ambiente do arquivo SD1
```

Função Posicione

Podemos também buscar uma informação em determinado campo usando apenas uma função.

Sintaxe:

Posicione(cAlias, nOrdem, cChave, cCampo)

Exemplo:

```
Posicione("SB1", 1, xFilial("SB1") + cCodigo, "B1_DESC")
```

Desta forma, será efetuada uma busca no SB1, na ordem 1, chave da busca xFilial("SB1") + cCodigo e será retornado o conteúdo do campo "B1_DESC". Note que esta função, não restaura a posição original do arquivo alvo (no caso SB1).

É necessário colocar a FILIAL do arquivo na chave passada como parâmetro, caso ela exista na chave do índice.

Função Existcpo

Retorna se determinada chave existe ou não no arquivo.

Sintaxe :

ExistCpo(cAlias,cChave,nOrdem)

Exemplo :

```
ExistCpo("SB1", 1, cCodigo, "B1_DESC")
```

Desta forma, será efetuada uma busca no SB1, na ordem 1, chave cChave. E será retornado se a chave foi encontrada ou não (.T. ou .F.). Neste caso não é necessário passar a filial. Ela será inserida automaticamente na chave de pesquisa pela função.

Restaurando Índice e limpando filtros

Nos relatórios devemos analisar que a função "SetPrint", possibilita efetuar filtros , escolha da ordem e geração em disco ou impressora , no final dos programas de relatório devemos restaurar a ordem original do arquivos e limpar o filtro e desativar a impressora.

```
//Término do relatório  
  
dbSelectArea("SRA")  
  
Set Filter to  
  
dbSetOrder(1)  
  
Set Device To Screen  
  
If aReturn[5] = 1  
    Set Printer To
```

```
Commit  
  
    curspool(wnrel)  
  
Endif  
  
MS_FLUSH()
```

Outros Cuidados

Um dos cuidados que devemos ter quando da criação de relatórios contendo valores é a utilização dos subtotais e totais, a fim de evitar erros que podem ser desastrosos durante uma tomada de decisão errada devido a valores errados.

A utilização de somatórias deve ser bastante criteriosa a fim de não cometermos o erro de misturarmos unidades de medidas diferentes no mesmo cálculo.

Confrontando relatórios e consultas

Quando elaboramos um sistema, existem muitos relatórios que geram dados para outros relatórios e consultas.

Devemos tomar cuidado para que não aconteçam divergências de informações de um para o outro, como por exemplo, no caso de valores.

Um bom exemplo disso, é a rotina de impressão de folha de pagamento. Este relatório exhibe informações que são utilizadas em outros relatórios, tais como, valores para o FGTS, guia de recolhimento de impostos.

Uma solução para que não se ocorra uma divergência de valores, seria utilizar uma única função ou rotina de processamento. Isto evitaria que ao se alterar o sistema, por motivo de lei ou outro qualquer, o programador alterasse por exemplo às rotinas de relatório de folha de pagamento e guia de impostos e esquecesse de alterar por exemplo à rotina de relatório de FGTS.

Exemplos como Saldos Bancários, Quantidades de Estoques, Valores de Faturamento, entre outros, devem ser confrontados entre relatórios e consultas para não gerarem informações errôneas ao cliente.

Normalmente estes problemas ocorrem em funções de critérios de filtragens diferenciados entre eles. Para evitar este tipo de problema é fundamental que o analista ao efetuar alguma manutenção em algum relatório ou consulta atente-se ao fato de assegurar que esta alteração não influencie outras situações.

Este é um tipo de não conformidade simples de ser evitada e que pode causar problemas sérios para os usuários além de ser de difícil argumentação quando nos questionado, pois evidencia falta de atenção ou critério na manutenção ou falta de conhecimento sobre o funcionamento do sistema.

Problemas com Looping de Programas

O Protheus utiliza a tecnologia Cliente/Servidor. Isto significa que o aplicativo não é mais executado individualmente em cada máquina, ele será executado no servidor do aplicativo. Até a versão 4.07 um programa travado significava que apenas a estação estava comprometida (o executável estava na memória da estação). Com o Protheus, todo o processamento está no Server e quando o programa está em looping estaremos gradativamente "usando toda a CPU do Server" e conseqüentemente parando todo o processamento.

Se ao desenvolvermos uma rotina e a mesma entrar em looping (tiver apenas uma entrada e não tiver uma saída do processamento), este processamento utilizará todos os recursos do servidor comprometendo (reduzindo drasticamente a performance do aplicativo), ou até impedindo, o uso do aplicativo por todos os demais usuários.

Se isso acontecer em uma empresa onde existem apenas 5 usuários, o administrador da rede poderá reiniciar o servidor, porém onde existe um número considerável de usuários poderá haver um prejuízo para a empresa que utiliza nosso sistema.

Exemplo:

```
dbSeek(xFilial("SE1")+DTOS(dDtIni))
```

```
Do While SE1->(!Eof())
```

```
...
```

```
...
```

```
<----- Faltava um DbSkip()
```

```
Enddo
```

No exemplo acima, a rotina ficará em looping (pois falta um comando de saída da rotina, um DbSkip() seria o mais apropriado), utilizando todos os recursos de processamento do servidor, fazendo com que o mesmo pare de funcionar.

Outro exemplo:

```
aCampos := {}
```

```
Do while .T.
```

```
    Aadd(aCampos, "Teste")
```

```
Enddo
```

No exemplo acima o caso é ainda mais crítico, pois além utilizar todo o recurso de processamento do servidor, em dado momento haverá uma queda do aplicativo, devido a limitação da variável tipo Array, criada acima. E quando este limite for ultrapassado, o sistema será interrompido abruptamente e todos os demais usuários ficarão impossibilitados de utilizarem o sistema.

Manipulação de Arquivos Externos ao Protheus

A manipulação de arquivos considerados externos ao Protheus deverá ter um tratamento diferenciado. Os arquivos a serem manipulados (alterados/consultados) deverão ser copiados do Client para o Server e vice-versa utilizando uma conexão (TPC-IP, IPX, etc). Para copiar os arquivos, foram criadas duas funções que serão executadas via conexão, a CPYS2T() encarregada de copiar do Server para o Client/Terminal e a CPYT2S() encarregada de copiar do Client/Terminal para o Server.

O editor de texto Word da Microsoft, os arquivos de imagens (BMP, JPEG, etc) exigem um lugar físico para abertura dos documentos/imagens, navegando pela Internet por exemplo são copiados via conexão para um diretório temporário no computador para serem visualizados.

O AP5 trabalha da mesma forma, através dessas considerações e utilizando a arquitetura Client/Server via conexão os arquivos serão copiados.

Em alguns Módulos do Protheus são encontradas rotinas de Importação/Exportação de lançamentos, exigindo serem utilizadas as funções CPYT2S() e CPYS2T() para manipulação dos arquivos. Por exemplo, uma importação de lançamentos da Folha de Pagamento poderá ser feita diretamente do Client sem precisar copiar para o Server mas se outro usuário precisar visualizar os lançamentos de origem da importação não terá acesso, agora se for realizado a cópia do Client para o Server todos poderão visualizar (aconselhável). Isso acontece no Módulo de Controle de Documentos, quando todos os arquivos (documentos) são copiados entre o Client e o Server para que todos visualizem e manipulem. Um exemplo que não há necessidade de cópia são os arquivos gerados para contabilização (CPROVA), pois estes são gerados no próprio Server não havendo necessidade de cópia.

Os arquivos que poderão ser copiados deverão estar necessariamente embaixo do RootPath na configuração do Server, isto é, o diretório DOCS do exemplo abaixo deverá ser sub-diretório do RootPath.

Exemplo de cópia do Server para o Client:

```
CPYS2T("\DOCS\EXEMPLO.DOC","C:\WINDOWS\TEMP",.T.)
```

Onde os parâmetros são:

- 1o. é o <Nome do Arquivo> a ser copiado para o Client
- 2o. é o <Nome do Diretório> do Client e/ou local físico onde será copiado o arquivo.
- 3o. se deseja compactar o arquivo (recomendável)

Exemplo de cópia do Client para o Server:

```
CPYT2S("C:\WINDOWS\TEMP\EXEMPLO.DOC","\DOCS",.T.)
```

Onde os parâmetros são:

- 1o. é o <Nome do Arquivo> a ser copiado para o Server
- 2o. é o <Nome do Diretório> do Server
- 3o. se deseja compactar o arquivo (recomendável)

As funções possuem um retorno True(.T.) ou False(.F.) indicando se a cópia foi realizada com sucesso ou não.

Desenvolvendo Telas

A aparência e objetividade das telas num sistema é base fundamental da interface Sistema x Usuário.

O AP5 já cria, automaticamente, a grande parte das telas de um módulo, tais como a Browse, a GetDados e Enchoice.

Algumas outras telas necessitam de construção "manual", ou seja, com a utilização de comandos, tais como "SAY", "GET" e "LABEL", na Dialog.

Procure sempre colocar em tela as informações que mais se objetivam com o assunto abordado.

Sempre que possível, dê preferência aos campos obrigatórios primeiro. Isso facilita a digitação do usuário, que não precisará passar de campo em campo (no caso de estar utilizando a tecla <TAB>) até chegar ao campo desejado. A ordem dos campos também é importante para a fácil localização das informações.

Quando o volume de informações é muito grande, divida os campos em folders, ou seja, pastas, agrupando os campos em assuntos. Isso irá deixar a tela menos poluída e evitará que o usuário navegue por uma tela só. Para fazer essa facilidade, preencha o campo X3_FOLDER, no SX3, com um número, agrupando-os de acordo com a tipo de informação e no SXA, com o ALIAS do arquivo em pauta, a ordem, que equivale ao número informado no X3_FOLDER e a descrição nos três idiomas. Essa descrição que será a informação contida na pasta do folder. Exemplo: Os campos SZ1_ENDER, SZ1_NUM e SZ1_BAIRRO devem estar com o campo X3_FOLDER preenchido com o conteúdo "1". No SXA, o XA_ALIAS deverá ser SZ1, o XA_ORDEM = "1" (mesmo valor preenchido no X3_FOLDER), no XA_DESCRIC, "Endereço Residencial" e, nos demais, o mesmo texto em outros idiomas.

O Folder, além de agrupar e facilitar a procura pelos campos, evita a rolagem vertical da tela, facilitando a visualização das informações.

Evite tela com muitos botões. Isso poderá confundir o usuário e induzi-lo ao erro. Utilize telas sequenciais, conhecidas como Wizard (semelhante aos de instalação de um software). Dessa forma, o usuário ficará

mais atento aos fatos, dificultando o erro. Mas cuidado: não faça disso uma incansável sequência de telas, pois isso acabará desmotivando o usuário a utilizar o sistema.

Enfim, as telas devem ser limpas e objetivas, de tal forma que impeça o usuário de sair de seu objetivo final. Todo curioso irá apertar todos os botões da tela ou preencher todos os campos com qualquer tipo de informação. Portanto, esteja atento a tamanho dos labels, para que os mesmos não excedam o tamanho da caixa de diálogo definida. Isso, além de não ser estético, prejudica o entendimento da informação.

Salvando Array's padrões

Quando temos Janelas que necessitem apresentar mais de uma getdados, devemos salvar os elementos, acols, aheader e n, da tela anterior para apresentar uma nova janela.

As principais variáveis são:

Acols = Array contendo as linhas usada que serão apresentadas na

Getdados

AHeader = Array contendo o cabeção das colunas da Getdados

N = Variável publica que indica a posição do atual no acols

(a Linha que está sendo editada na Getdados)

Para salva-las podemos:

aColsAnt := aClone(Acols)

aHeaderAnt := aClone(aHeader)

nElemAnt := n

E para restaura-las:

aCols := aClone(aColsAnt)

aHeader := aClone(aHeaderAnt)

n := nElemAnt

Pontos de Entrada

Dentro dos processos operacionais dos programas é possível criar "aberturas" que possibilitam a execução de processos distintos a partir de uma rotina do sistema.

Objetivo dos Pontos de Entrada

Deixar o sistema flexível, pois isso permite uma grande variedade de desenvolvimento pelos nossos analistas de suporte de acordo com a necessidade de cada tipo de cliente/implantação.

Permitir que o sistema seja o mais abrangente possível de acordo com cada tipo de segmento de negócio.

Quando criar um Ponto de Entrada

O ponto de entrada tem resultado quando permite que um processo possa ser modificado, desde que ele não seja crítico para o sistema.

Exemplo: Montagem das parcelas de pagamento de um pedido

Ele é útil em processos que podem ser diferentes de acordo com o tipo de negócio de cada empresa ou estratégia adotada.

Ex: Relatório de Pedido, Cadastro de Clientes

Sintaxe para criar um Ponto de Entrada:

```
Function TMKA010()
```

```
Local IRetorno := .F.
```

```
Local LTMKMCL := Existblock("TMKMCI") // O "Existblock" detecta se existe uma função no repositório do AP5 com esse PE.
```

```
If LTMKMCL
```

```
    IRetorno := Execblock("TMKMCI", .F., F., {aValor}) // aValor é um array que seria recebido pelo usuario em PARAMIXB
```

```
Endif
```

```
Return( IRetorno )
```

Procedimentos para sua criação

Avaliar com critério a criação do Ponto de Entrada, pois é importante localiza-lo num ponto que seja útil, não redudante e que realmente dê condições de atender ao solicitante.

O Ponto de entrada não é uma ferramenta de correção de eventuais falha do sistema e sim para ajudar no desenvolvimento de negócios específicos.

Documentar no QUARK com nome, momento no qual o momento em que ele é disparado, parametros que ele envia, retorno esperado (se houver) e o que ele executa com o retorno.

Contabilizando

A contabilização de registros em um sistema ERP é necessário toda vez que houverem operações envolvendo valores na empresa, como por exemplo, a emissão de uma nota fiscal, um recebimento de dinheiro, um pagamento na folha, entre outros.

Para quem está programando é fundamental que este conceito esteja bastante sidimentado.

Para se definir a contabilização em qualquer rotina deve-se :

- Definir os lançamentos padronizados que serão utilizados para a rotina (cada módulo ou família de módulo possui uma sequencia lógica)
- Preparar o programa para efetuar as atualizações e a contabilização. Este detalhe é muito importante pois o lançamento padrão é uma fórmula e o posicionamento dos registros é fundamental. Exemplo : Na rotina de exclusao de nota fiscal deve-se executar a contabilização antes do comando dbDelete().
- A chamada do lançamento contábil deve estar em um lugar estrategicamente correto, pois teoricamente é a última coisa a ser feita na rotina. Não é aconselhável executar a contabilização e depois efetuar outros comandos.

Existem três funções que deverão ser utilizadas, para que seja criado um arquivo texto, contendo as informações a serem contabilizadas.

Como temos 2 sistemas contábeis (SIGACON e SIGACTB) com procedimentos diferenciados programamos de forma idêntica para ambos porém internamente as rotinas efetuam comandos diferentes. Os comandos para a contabilização são :

A Função HeadProva

Esta função cria o cabeçalho da contabilização. É tratada de forma diferenciada para os módulos SIGACON e SIGACTB.

Sintaxe:

`nHdlPrv := HeadProva(cLoteAtf, cNomProg, Substr(cUsuario,7,6), @arquivo, .T.)`

Onde:

`nHdlPrv` -> Variável que conterá o num. (Handle) do arquivo (.LAN) criado.

`cLoteAtf` -> Código do lote do módulo (Ex.: Ativo Fixo: "8866")

`cNomProg` -> Nome do Programa (Ex.: "ATFA060")

`cUsuario` -> Usuário arquivo: nome do arquivo (Ex.: `cArquivo := ' '`)

A função DetProva()

Em primeiro lugar, deve-se estar posicionado no registro, que contém o valor a ser contabilizado

Sintaxe:

`ExpN1 := DetProva(ExpN2,ExpC1,ExpC2,ExpC3)`

`ExpN1` -> Valor Total da Contabilização

`ExpN2` -> Handle retornado da função anterior

`ExpC1` -> Código do Lançamento Padrão

`ExpC2` -> Nome da rotina Geradora

`ExpC3` -> Lançamento Padrão

A função RodaProva()

Esta função irá criar a finalização da contabilização.

Sintaxe:

`RodaProva(ExpN1, ExpN2)`

`ExpN1` -> Handle retornado da função anterior

`ExpN2` -> Valor Total da contabilização

No final, ou seja, após todos os registros serem processados utilizar a função `CA100INCL()`, cujo objetivo é ler o arquivo gerado (.LAN), e gerar os lançamentos no arquivo SI2 (Lançamentos contábeis).

Exemplo:

`CA100Incl(cArquivo, nHdlPrv, nOpcx, cLoteContabil, lDigita, lAglut, cOnLine, dData)`

Onde:

cArquivo -> Nome do arquivo

nHdlPrv -> Numero do Header

nOpcx -> Numero da Opcao escolhida

cLoteContabil -> Numero do Lote

IDigita -> Se Mostra ou nao

IAglut -> Se Aglutina ou não

cOnLine -> Determina se sera On Line ou pelo cProva

Atualizando SX (ATUSX)

O ATUSX é uma ferramenta muito importante utilizada na manutenção dos arquivos customizadores internos: os arquivos SX's . É nos arquivos SX que estão armazenadas todas as informações padrões necessárias para a criação de bases das empresas e efetuar possíveis customizações.

Em uma atualização de versão, o sistema irá fazer uma compatibilização de dados com os arquivos já existentes e os atualizará com base nos SX's. Ao desenvolver uma rotina que exige um novo campo, por exemplo, este deverá ser criado no ATUSX, caso contrário gerará uma não conformidade comprometendo todo o trabalho da atualização e o bom andamento do trabalho. Todos os campos, perguntas, índices, parâmetros novos deverão estar no ATUSX, pois estes são os padrões do Protheus.

É aqui também que cadastramos os HELP's de campo e de programas, criamos os menus do sistema e onde disponibilizamos informações para serem traduzidas para outros países.

Exemplo de atualização no SX:

Criação de Índice no Cadastro de Funcionários- Suponhamos que seja necessário um índice por Data de Admissão. Neste caso utilizaremos o SINDEIX, onde deverão ser alimentados basicamente o Alias (SRA), a ordem, a chave de indexação (RA_ADMISSA) e sua descrição em Português. As descrições referentes às outras línguas deverão ficar a cargo do departamento de traduções. Numa atualização de versão, o sistema enxergará a existência deste índice pelo SINDEIX e o disponibilizará para utilização.

Ao final da manutenção dos arquivos SX's, abre-se uma janela onde deve ser documentado todas as alterações efetuadas da forma mais clara e precisa possível. Esta documentação é de extrema importância para que se tenha um controle dos arquivos customizadores padrões e garantem um perfeito funcionamento do Protheus.

SX1 - Parâmetros Genéricos

Esta tabela contém as perguntas, os valores armazenados e a última resposta utilizada para processamento, impressão de relatórios, etc.

Todos os processamentos ou relatórios que tenham a opção de parametrização, deverão utilizar a função Pergunte para carregar os valores dos parâmetros e/ou apresentar na tela as perguntas relacionadas.

Como o usuário pode não utilizar a opção de parametrização, devemos sempre carregar as variáveis MV_PARXX com os valores default ou com as últimas respostas aplicadas, para evitar erros de comparação de variáveis.

SX2 - Mapeamento dos arquivos

Armazena os Paths dos arquivos possibilitando que estes possam ser distribuídos em diversos drives ou diretórios.

SX3 - Dicionário de Dados

Armazena informações referentes às definições de todos os campos que existem no Protheus.

SX5 - Tabelas Genéricas

Armazena tabelas genéricas utilizadas em todo sistema.

SX6 - Parâmetros

Elemento chave para a execução de um processamento, que determina diferentes resultados dependendo do seu conteúdo.

SX7 – Gatilhos

Rotina ou operação que é disparada a partir de um evento get.

SXE/SXF - Seqüência. de documentos

Armazena sequencias alfanuméricas que deverão ser controladas por um semáforo. Um exemplo clássico seria a numeração de pedidos de venda, em que vários usuários poderiam estar utilizando o mesmo número de pedido.

SINDEX- Arquivo de Índices

Armazena todos os índices padrões.

Controle de Transação (TTS)

O que é

Tratando-se de Banco de Dados, toda e qualquer operação de inclusão, alteração ou exclusão de registro é armazenada primeiramente na área de LOG, garantindo assim que ao fazer a inclusão de uma linha (registro) seja garantida a inclusão completa de todas as colunas (campos). Caso não seja possível a inclusão da linha completa ele executa um procedimento chamado de ROLLBACK, ou seja, ignora todo o registro.

Quando usar

Quando temos uma operação em Banco de Dados que necessite que várias inclusões, alterações ou exclusões só sejam efetuadas quando todas as operações tenham sido realizadas com sucesso, garantindo com isso que não seja atualizada parcialmente uma tabela ou que atualize uma tabela e não atualize outra tabela relacionada.

Como usar

Para definir uma transação, deve-se utilizar os comandos BEGIN TRANSACTION e END TRANSACTION para definir início e fim de uma transação respectivamente. Todas informações a serem gravadas no Banco devem estar dentro de uma única transação sejam elas provenientes de uma ou várias tabelas.

Deve-se evitar utilizar laços (WHILE, FOR) dentro de uma transação, pois a área de LOG do banco é limitada, e se o volume de informações ultrapassarem este limite, ocasionará o travamento do banco de dados. O tamanho da transação deve ser conhecido pelo programador. Em suma, para exemplificar, devemos controlar a transação de uma nota e não de um conjunto ilimitado de notas.

Onde não usar

O controle de transação jamais deverá ser utilizado durante processo que envolvam interface (telas com entrada de dados). O controle deve-se resumir apenas ao processo de gravação. Entre um início de transação (Begin Transaction) e um final (End Transaction) Todos os registros a serem gravados ficam "locados" até o final da transação. Caso tenhamos uma tela após o BEGIN e antes do END dependeremos do usuário para efetuar a liberação da transação, fato este que causaria enormes problemas para o usuário.

Outro lugar que não deve-se ter o controle de transação refere-se a rotinas de reprocessamentos ou recálculos, onde as informações podem ser regeados durante este processo ou onde possamos ter um grande número de locks.

BEGIN TRANSACTION

ExpN1 :=FuncGrava()

END TRANSACTION

Caso exista uma transação dentro de uma outra a segunda será automaticamente ignorada, fechando-se a transação principal quando da chamada do comando END TRANSACTION.

Comandos Definidos pelo Usuário (UDC's)

Este recurso cria novas e infinitas possibilidades para modificar a maneira pela qual escrevemos o código de uma função e a maneira pela qual podemos resolver problemas complexos. Ele ajuda a facilitar a manutenção do código, e a implementação de normas.

Estes comandos são traduzidos, analisados e modificados antes que o compilador comece a trabalhar para gerar um arquivo objeto. O responsável por esta tradução é o pré-processador que é um tradutor inteligente que atua antes da geração do código objeto.

Em sua maioria, isto se resume a encontrar os comandos no código fonte e traduzi-los para instruções e funções equivalentes que se acham no corpo da função ou no conteúdo de arquivos .CH (arquivos de cabeçalho). Este tipo de arquivo (.CH), contém diversos comandos que serão utilizados por todas as funções que contenham a instrução "include" em seu código.

Estes mesmos comandos poderiam estar embutidos na função, mas para facilitar a manutenção, um único arquivo .CH, pode ser incluído (comando include) em várias funções ao mesmo tempo. Não há a necessidade de colocar o comando include em cada função. Uma única menção ao .CH no arquivo .PR?, servirá ao propósito de todas as funções nele embutidas.

Estes comandos são diretivas do pré-processador e começam sempre com o caracter "#" diretamente à sua frente e devem estar escritos em caracteres tipo caixa alta. Os mais utilizados no Protheus são:

#DEFINE

#IFDEF, ou #IFNDEF

#ELSE

```
#ENDIF
```

```
#INCLUDE
```

Cada um destes comandos pode ser colocado em qualquer parte do arquivo fonte, que será lido pelo pré-processador. No entanto, para facilitar a visualização da existência destes comandos e manutenção da função, estes comandos devem ser colocados no início do fonte.

O pré-processador substituirá, cada ocorrência da constante no fonte, será substituída pela expressão ou valor, que estiver contida diretamente à frente da mesma. A exemplo de sintaxe e código fonte para este comando é:

```
#DEFINE _TESC 27
```

```
#DEFINE _LESC lastkey()
```

```
if _nLastkey == _TESC
```

```
    RETURN
```

```
Endif
```

```
if _LESC == _TESC
```

```
    RETURN
```

```
Endif
```

Esta diretiva é muito útil quando temos um valor constante várias vezes repetido dentro do código fonte, que poderá ser alterado com frequência no decorrer da vida útil da função.

```
#IFDEF ou #IFNDEF <CONSTANTE>
```

```
    <instruções>
```

```
#ELSE
```

```
    <outras instruções>
```

```
#ENDIF
```

Esta diretiva do pré-processador permite que você prepare aplicações para compilação condicional, ou em outras palavras, se a <constante>, definida anteriormente via diretiva DEFINE ou pelo Protheus foi definida (IFDEF), as <instruções> serão incorporadas ao código objeto e as <outras instruções>, serão desprezadas. Mas se a <constante> não foi definida <IFNDEF> as <instruções> serão incorporadas e as <outras instruções> serão desprezadas.

```
#INCLUDE "<ARQUIVO>"
```

Uma instrução INCLUDE diz ao pré-processador para inserir o conteúdo de outro arquivo em um local determinado dentro da função. Este arquivo especificado, deve focalizar outras instruções do pré-processador que possam ser comuns a diversos módulos da rotina.

Uso de Strings

Para que o sistema possa ser utilizado em outros países com línguas diferentes ao invés de inserirmos diretamente os textos no fonte do relatório utilizamos o recurso de strings, onde através de um include acessaremos o texto em três línguas diferentes (Português, Espanhol e Inglês) dependendo da forma da compilação.

É convencionalizado que o nome do include seja o mesmo nome do fonte do relatório para que a manutenção deste relatório seja o mais simples possível.

Exemplo:

Relatório -> FABR001.PRW

Include -> FABR001.CH

Conceito de Filial e Compartilhamento de Arquivos

O Sistema permite a criação de várias Filiais para uma mesma empresa cadastrada, de modo que Filiais da mesma empresa compartilhem ou não as mesmas informações como Cadastro de Produtos, Clientes, etc.

Este tratamento é feito internamente através dos campo XX_FILIAL obedecendo a seguinte regra:

Arquivos Compartilhados

Quando o arquivo esta configurado para trabalhar no modo compartilhado (X2_MODALO = 'C'), este campo será gravado com " " (espaços).

Deste modo o registro ficara disponível para todas as Filiais.

Arquivos Exclusivos

Quando o arquivo esta configurado para trabalhar no modo exclusivo (X2_MODALO= 'E'), esta campo será gravado com o código da Filial Atual.

Deste modo o registro focara disponível apenas para a Filial que o gravou.

Para que o registro realmente fique disponível ou não para suas respectivas Filiais , TODAS as rotinas que manipulam registros diretamente na base de dados deverá verificar a Filial através da Função xFilial() , alem disto a maioria dos índices possuem o campo FILIAL na

chave :

Sintaxe : XFILIAL(EXPC1) onde, ExpC1 = Alias do arquivo

A função xFilial() verifica se o arquivo é exclusivo ou compartilhado e ira retornar " " se o arquivo for Compartilhado e o código da Filial se o arquivo for exclusivo .

Por exemplo :

Para executar um dbSeek no arquivo de clientes :

DbSelectArea("SA1")

DbSeek(xFilial("SA1")+cCodCli+cLoja)

Índice do SA1 :

A1_FILIAL+A1_COD+A1_LOJA

Ou um processamento no arquivo :

Do while !EOF() .AND. XX_FILIAL=xFilial("01")

Sendo o campo FILIAL parte da chave de Todos os índices do sistema, este procedimento garante que a utilização dos registros será exclusiva da Filial que criou os mesmos no caso do arquivo ser Exclusivo, ou disponível para todas as Filiais quando o mesmo estiver configurado como Compartilhado.

Jamais use um campo filial de uma tabela para executar um `dbSeek()` em outra tabela. Pois uma tabela poderá ser compartilhada (campo filial em branco), enquanto que a outra poderá ser compartilhada (campo filial preenchido).

A variável `cFilAnt` contém a filial que o usuário está operando, e a variável `cEmpant` contém a empresa e a filial

Técnicas para Filtragem

Nos sistemas Microsiga, a filtragem dos dados em ambiente Code Base os "DBFs" é feita de maneira geral pela Função `INDREGUA(Params,)`, o que de forma geral não impede o uso de instruções como `SET FILTER TO`, `DBSETFILTER()`, `DBFILTER()` ou qualquer outro comando de sintaxe xBase. A `INDREGUA()` é uma função interna que reúne vários atrativos e facilidades para o seu uso, entre elas a possibilidade de se indexar e filtrar os registros através dos parâmetros fornecidos, ela também é preferencialmente usada para que o código fique mais "limpo" e de fácil interpretação pois dentro de sua estrutura reúne uma série de comandos de indexação e filtragem que agiliza o processo de criação de índices e filtros em arquivos de trabalho com menos linhas de código, vejamos o Exemplo :

Chaves Primárias

Sua função é garantir unicidade. Em toda relação, por definição, tem-se uma ou mais chaves candidatas. Dessas chaves, uma será primária e se houver mais de uma na relação, essas outras serão definidas como chave alternada.

Chaves Estrangeiras

É um atributo cuja função é permitir relacionamento. Em uma tabela na qual o atributo é chave externa ou estrangeira, em outra, o atributo deve ser chave primária, e os valores dos campos são necessários.

Integridade Referencial

Todos os valores da chave estrangeira tem, obrigatoriamente, que ter valor correspondente na chave primária que se relaciona; mas nem todos os valores encontrados na chave primária, precisam ter seus correspondentes na chave estrangeira que se relaciona. Por exemplo, na tabela de clientes, o campo `A1_COD` (código do cliente), vai estar relacionado com outra tabela que indica quais são os pedidos de venda colocados. Desta forma, nem todos os clientes precisam ter pedidos de venda colocados; mas, necessariamente, todos os pedidos de venda precisam de um cliente.

Como o PROTHEUS foi projetado para o ambiente SQL, onde a integridade referencial das tabelas é definida no próprio banco de dados através de regras internas, devemos tomar algumas precauções com esse tópico:

Verificar a integridade da coluna em todas as tabelas relacionadas: não pode-se alterar o tamanho do código do cliente em apenas uma tabela, caso esse código seja alterado deve-se verificar as tabelas de cabeçalho e itens das notas fiscais, de títulos a pagar e receber, etc. O sistema conta com o recurso de grupos de tabelas relacionadas, que permite alterar o tamanho de diversas colunas de uma vez só, garantindo a integridade das colunas

Verificar a integridade dos cadastros com todas as tabelas relacionadas: não pode-se excluir o código do cliente se existe um pedido de vendas em aberto para esse cliente, deve-se verificar todas as tabelas relacionadas antes de atualizar a base de dados. Além disso na inclusão de cadastros devemos utilizar as funções `existchav` e `existcpo` para garantir que as informações de chave não sejam repetidas e que o acesso a tabelas externas seja validado de maneira consistente.

Verificar a atualização da informação em todas as tabelas relacionadas: a integridade não se resume a validações de cadastros e tamanho de colunas, deve-se garantir no ato do desenvolvimento que TODOS os pontos relacionados ao tópico envolvido sejam analisados e se necessário atualizados. Por exemplo, se será atualizado o saldo em estoque de determinado produto NÃO DEVE-SE atualizar somente o arquivo de saldos em estoque, deve-se avaliar se o produto utiliza rastreabilidade para nesse caso atualizar o arquivo de saldos por lote, deve-se avaliar se o produto utiliza controle de localização física para nesse caso

atualizar o arquivo de saldos por localização, etc. Deve-se fazer um estudo antes de qualquer alteração em atualização de base de dados.

Utilizando Rotinas Automáticas

A cada dia estamos criando rotinas com interface automática para melhorar a entrada de dados via outros equipamentos, tais como coletores de dados, interface de outros softwares, etc. Porém, para nossa própria portabilidade e utilização de rotinas padronizadas, temos adotado o próprio programa standard, contudo sem interferência do usuário (digitador). Para tal, criamos um mecanismo onde todos os programas que necessitem desta regra devem ser capazes de "inserir" dados de forma automática. Abaixo mostraremos como proceder :

Tome como exemplo o MATA250.PRX . O vetor aRotAuto é passado para o programa citado. Se este vetor contiver elementos, significa que será utilizada a Rotina Automática. Este vetor deve, quando da utilização das rotinas automáticas, conter os dados mínimos necessários para a atualização dos arquivos.

Veja a estrutura do vetor a ser enviado para a rotina automática.

aRotAuto := { cCampo, Conteúdo, Validação }

Onde

cCampo -> é o campo a ser atualizado,

Conteúdo -> é o conteúdo que cCampo vai receber

Validação -> é a validação que cCampo vai receber.

Observação: A Validação pode ser uma função ou um valor 'NIL'. Se for 'NIL', as validações a serem utilizadas para o respectivo campo serão as existentes no SX3. Se as validações não forem as do SX3, elas devem ser passadas numa função.

Exemplo,

```
aRotAuto := { { "D3_TM" ,"001" ,NIL } , ;  
{ "D3_COD" ,padr("10100",15) ,NIL } , ;  
{ "D3_UM" ,"UN" ,NIL } , ;  
{ "D3_QUANT" ,1 ,NIL } , ;  
{ "D3_OP" ,"00000401001" ,NIL } , ;  
{ "D3_LOCAL" ,"01" ,NIL } , ;  
{ "D3_EMISSAO" ,dDataBase ,NIL } }
```

Para o processo de inclusão simples, sem getdados, a variável padrão a ser utilizada nos programas chama-se aRotAuto, e para processo de inclusão com cabeçalho e itens, as variáveis a serem utilizadas são: aAutoCab para o cabeçalho, e aAutoItens para os itens da getdados.

Para uma inclusão simples, tomar como exemplo o MATA250.PRX. Para uma inclusão com cabeçalho e item, tomar como exemplo o CONA050.PRX.

Controle de Semáforo

O controle de Semaforo permite que o sistema controle a Numeração Automática de Documentos On Line. Temos basicamente 3 funções que gerenciam o controle do mesmo.

São elas :

GETSXENUM(EXPC1) -> Obtem o número sequencial do alias especificado no parâmetro.

ROLLBACKSXE -> Descarta o número pendente do semáforo. É usado quando o usuário cancela a operação (o numero não é aproveitado).

CONFIRMSXE -> Confirma o número sugerido. Esta função deve ser chamada quando da confirmação da gravação do registro.

MAYIUSE -> Checa fisicamente se um determinado arquivo existe. O arquivo poderá conter o número sequencial.

Obs : A função GETX8NUM executa a própria GETSXENUM.

Atualização do SourceSafe

A atualização do Source Safe é a última, e mais crítica, etapa do processo de alteração dos programas. Para que ela seja feita com a máxima segurança algumas etapas devem ser observadas. A seguir:

Após as alterações será necessária uma análise meticulosa de tudo o que foi alterado para avaliar qual o impacto que estas poderão causar no programa em que foram executadas e nos programas correlacionados;

Deverão ser efetuados o maior número de testes onde deverão ser previstos o maior número de situações possíveis e prováveis. Os testes deverão ser simulados em base equivalente à instalada no cliente e para as novas versões, em uma nova base que poderá ser gerada através do ATUSX;

Feita a análise, os testes e antes de atualizar definitivamente o Source Safe o programa alterado deverá ser comparado com o constante no Source Safe para verificar se as alterações foram realmente feitas no programa que foi reservado e, só então o programa poderá ser "Baixado".

Após a "Baixa", e para garantir que a atualização do Source Safe foi feita corretamente, o programa atualizado deverá ser novamente comparado. E é claro que se todas as etapas anteriores, principalmente a 1 e 2, foram cumpridas essa não passará de uma medida de segurança.

Procedimentos de Localizações

A Microsiga atua hoje em 13 países e isto faz com que praticamente todas as alterações executadas no sistema reflitam em todos os países (exceto quando não for necessário tal procedimento).

Procedimentos a serem cumpridos em alteração / desenvolvimento de programas :

A fim de evitar os inconvenientes citados no ponto anterior, existem procedimentos que devem ser adotados no nosso dia a dia e repassado para aqueles funcionários que se acoplam a nossa equipe. Estes são (entre parêntesis os problemas que seriam reduzidos usando os procedimentos) :

- Quando é incluído um novo STR em um CH, ou criado um novo CH, ou modificado um STR em um CH já existente, este deve ser replicado em PORTUGUÊS para os demais idiomas e automaticamente deve ser encaminhado um e-mail para traducoes@microsiga.com.br indicando a versão, o STR e o CH que foi alterado.

- Quando criado um campo novo, ou modificado o conteúdo de um já existente, os campos que devem refletir esta alteração nos demais idiomas devem ser deixados em branco, assim é como o pessoal de traduções identifica os campos que devem ser traduzidos. Isto é válido para todos os arquivos do dicionário de dados.
- Quando criado ou alterado um novo HELP (de campo ou de programa) deve ser informado de imediato para traduções para proceder a tradução para os outros idiomas.
- Para atualizar um parâmetro deve ser sempre usada a função PUTMV, NUNCA DEVE SER PREENCHIDO NEM POSICIONADO POR FORA. Esta função atualiza nos três idiomas.
- Quando criado um campo novo "similar" a outros já existentes no sistema, deve se analisar se este deve ser replicado com características diferentes para todos os países localizados, ou se as características devem ser igual independentemente da localização. Na falta de "critério" ou informação, deve ser informado ao setor de localizações.
- Quando criado um campo novo de uso exclusivo de Brasil (E1_INSS por exemplo) deve ser informada a equipe de localizações para configurar este campo (uso, browse, etc.) de acordo com os demais países.
- Quando for modificada a característica de um campo do sistema e este estiver replicado para o resto dos países, as alterações devem ser replicadas em todos os países. Na dúvida da aplicabilidade da alteração nos outros países, deve ser informada a equipe de localizações.
- Os novos campos tipo COMBO, devem ser criados com numeração e não com siglas (1 para sim e 2 para não, ao invés de S para sim e N para não). Esta alteração o inclusão deve ser informada de imediato para a equipe de traduções.
- Quando for criado um novo parâmetro, ou modificado o conteúdo default de um já existente, esta modificação deve ser aplicada nas 3 línguas.
- Quando houve a possibilidade de pegar um STR do dicionário (função RETTITLE()), este deve ser pego, o que evita ter que criar vários STR e tratarmos com a variável cPaisLoc dentro do programa. Exemplo CGC, NOTA FISCAL, CEP, etc.
- Não deve ser usada a acentuação
- Quando criadas novas perguntas no SX1, ou modificadas as existentes, e o X1_GSC for igual a C, deve ser informado de imediato para traduções, pois os campos das opções do combo, quando vazias, geram erro fatal na execução.

Programando com Schedule de Relatórios

Como o sistema permite que a emissão de relatórios possa ser programada (schedule) é fundamental que se utilize as rotinas padrões para a emissão dos mesmo. O controle do schedule é feito pela função SetPrint. Sendo assim, não é suportado interface com data entry durante o processo de relatório, visto que isto inviabilizará a utilização do mesmo. A não ser em relatórios específicos e que sejam inviáveis a utilização de shedule (ex. Impressão de Cheques) este procedimento deverá ser adotado corretamente.

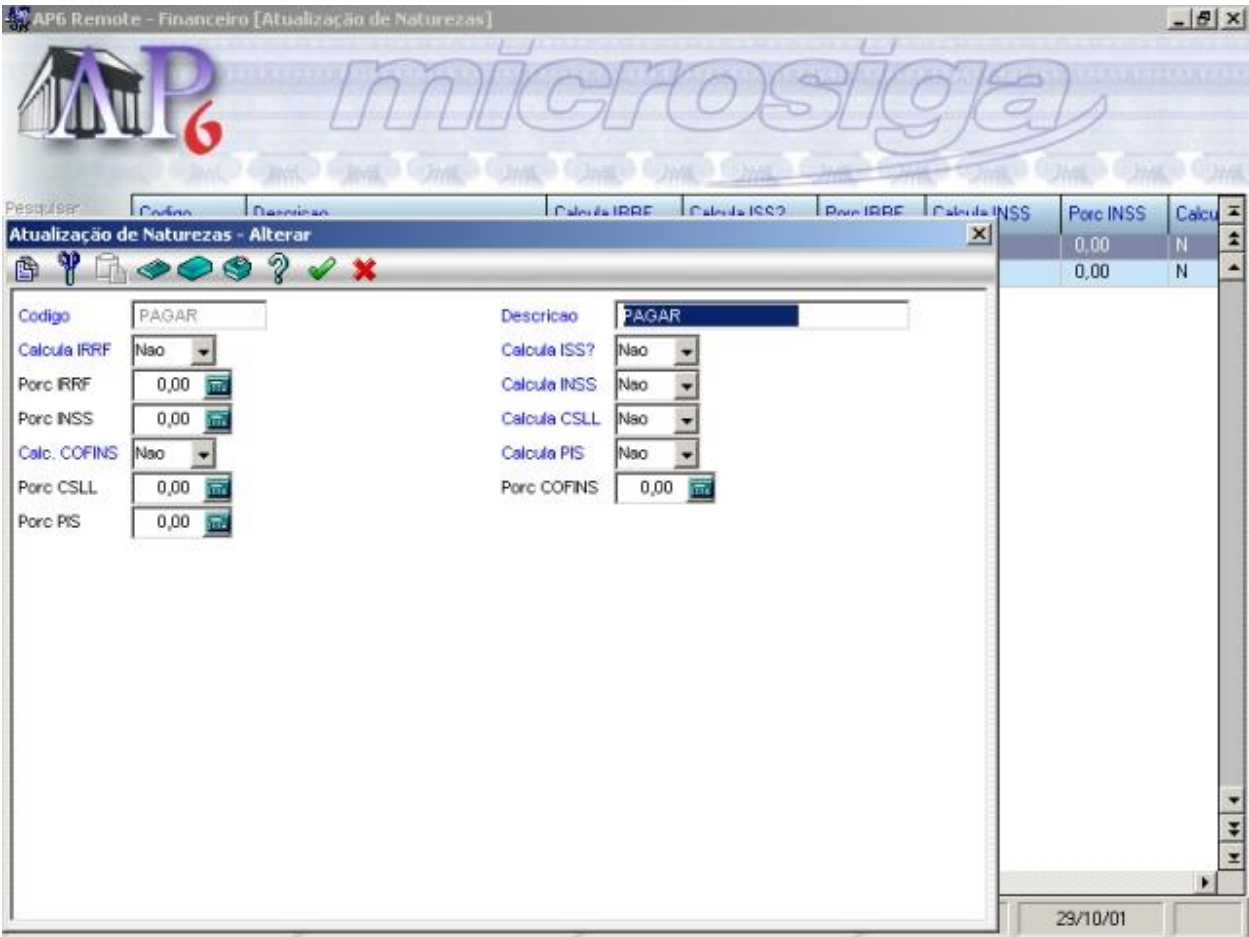
Caso exista alguma entrada de dados que seja possível ser assumida qualquer valor apenas no schedule deve-se adotar o seguinte procedimento :

Usar a variável __cInternet, que se estiver com valor .T. (Verdadeiro) estamos no processo de schedule.

Modelos de Programas Padronizados

Modelo 1

Este modelo de programa exibe um Browse vertical de campos presentes no dicionário de dados. Genericamente as validações são herdadas do próprio dicionário de dados.



```

/* /
+-----+
+ Funcao   | FINA010 | Autor | Wagner Xavier          | Data | 28/04/92 |
+-----+-----+-----+-----+
| Descricao | Programa de atualizacao de Naturezas |
+-----+-----+-----+-----+
| Sintaxe   | FINA010() |
+-----+-----+-----+-----+
| Uso       | Generico   |
+-----+-----+-----+-----+
|          | ATUALIZACOES SOFRIDAS DESDE A CONSTRUCAO NICIAL |
+-----+-----+-----+-----+
| Programador | Data   | BOPS | Motivo da Alteracao |
+-----+-----+-----+-----+
|          |          |          |          |
+-----+-----+-----+-----+
/* /

```

```

#include "FINA010.CH"
#include "PROTHEUS.CH"
FUNCTION FINA010
/*
+-----+
| Define Array contendo as Rotinas a executar do programa          +
| ----- Elementos contidos por dimensao -----                  +
| 1. Nome a aparecer no cabecalho                                   +
| 2. Nome da Rotina associada                                       +
| 3. Usado pela rotina                                             +
| 4. Tipo de Transacao a ser efetuada                               +
|   1 - Pesquisa e Posiciona em um Banco de Dados                 +
|   2 - Simplesmente Mostra os Campos                             +
|   3 - Inclui registros no Bancos de Dados                       +
|   4 - Altera o registro corrente                                 +
|   5 - Remove o registro corrente do Banco de Dados               +
+-----+
*/
PRIVATE aRotina := { { OemToAnsi(STR0001) ,"AxPesqui", 0 , 1},;      //"Pesquisar"
                    { OemToAnsi(STR0002) ,"AxVisual", 0 , 2},;      //"Visualizar"
                    { OemToAnsi(STR0003) ,"AxInclui", 0 , 3},;      //"Incluir"
                    { OemToAnsi(STR0004) ,"AxAltera", 0 , 4},;      //"Alterar"
                    { OemToAnsi(STR0005) ,"FA010Del", 0 , 5, 3} }    //"Excluir"
+-----+
| Define o cabecalho da tela de atualizacoes                       |
+-----+
PRIVATE cCadastro := OemToAnsi(STR0006) //"Atualizacao de Naturezas"
+-----+
| Endereca funcao Mbrowse                                          |
+-----+
mbrowse( 6, 1,22,75,"SED")
Return
/*
+-----+-----+-----+-----+-----+-----+
| Funcao      |FA010DEL | Autor | Wagner Xavier          | Data | 8/04/92 |
+-----+-----+-----+-----+-----+-----+
| Descricao   | Programa de exclusao de Naturezas
+-----+-----+-----+-----+-----+-----+
| Sintaxe     | A010Deleta(ExpC1,ExpN1,ExpN2)
+-----+-----+-----+-----+-----+-----+
| Parametros  | ExpC1 = Alias do arquivo
|              | ExpN1 = Numero do registro
|              | ExpN2 = Numero da opcao selecionada
+-----+-----+-----+-----+-----+-----+
| Uso         | FINA010
+-----+-----+-----+-----+-----+-----+
*/
FUNCTION FA010DEL(cAlias,nReg,nOpc)
Local aAC := { OemToAnsi(STR0007),OemToAnsi(STR0008) } //"Abandona"###"Confirma"
Local bCampo
Local lDeleta := .T.
Local oDlg
Local nCont
Local nOpcA
+-----+
| Monta a entrada de dados do arquivo                             |
+-----+
Private aTELA[0][0],aGETS[0]
+-----+
| Verifica se o arquivo esta realmente vazio ou se esta          |
| posicionado em outra filial                                     |
+-----+
If EOF() .or. SED->ED_FILIAL != xFilial("SED")
HELP(" " , 1 , "ARQVAZIO")
Return Nil
Endif

While .T.
+-----+
| Envia para processamento dos Gets                             |
+-----+

```

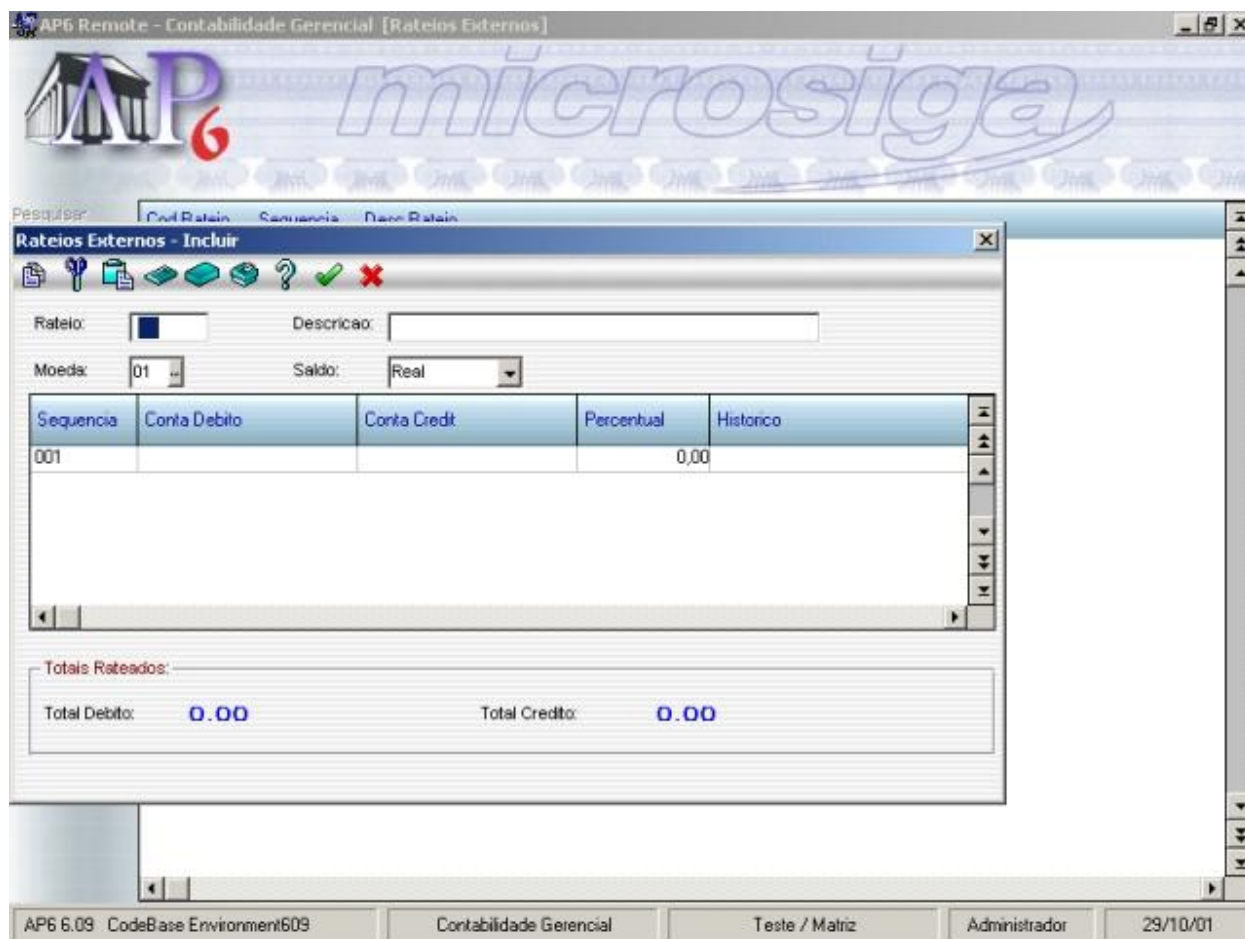
```

dbSelectArea( cAlias )
bCampo := { |nCPO| Field(nCPO) }
FOR nCont := 1 TO FCount()
    M->&(EVAL(bCampo,nCont)) := FieldGet(nCont)
NEXT nCont
nOpcA := 1
DEFINE MSDIALOG oDlg TITLE cCadastro FROM 9,0 TO 28,80 OF oMainWnd
EnChoice( cAlias, nReg, nOpc, , "AC", OemToAnsi(STR0009) ) //"Quanto a exclusao?"
ACTIVATE MSDIALOG oDlg ON INIT EnchoiceBar(oDlg, { || nOpcA := 2, oDlg:End() }, ;
{ || nOpcA := 1, oDlg:End() })
DbSelectArea(cAlias)
dbSelectArea(cAlias)
IF nOpcA == 2
    +-----+
    |   Antes de deletar, verificar se existe movimentacao   |
    +-----+
    dbSelectArea("SE1")
    dbSetOrder(3)
    IF (dbSeek(cFilial+SED->ED_CODIGO))
        Help(" ", 1, "A010NAODEL")
        lDeleta := .F.
        MsUnlock()
    Else
        dbSelectArea("SE2")
        dbSetOrder(2)
        IF (dbSeek(cFilial+SED->ED_CODIGO))
            Help(" ", 1, "A010NAODEL")
            lDeleta := .F.
            MsUnlock()
        Else
            dbSelectArea("SE5")
            dbSetOrder(4)
            IF (dbSeek(cFilial+SED->ED_CODIGO))
                Help(" ", 1, "A010NAODEL")
                lDeleta := .F.
                MsUnlock()
            Endif
        Endif
    Endif
    Endif
    If lDeleta
        +-----+
        |   Inicio da Protecao via TTS   |
        +-----+
        BEGIN TRANSACTION
        dbSelectArea(cAlias)
        RecLock(cAlias, .F., .T.)
        dbDelete()
        END TRANSACTION
        +-----+
        |   Final da protecao via TTS   |
        +-----+
    Endif
Else
    MsUnlock()
Endif
Exit
Enddo
dbSelectArea("SE1")
dbSetOrder(1)
dbSelectArea("SE2")
dbSetOrder(1)
dbSelectArea("SE5")
dbSetOrder(1)
dbSelectArea(cAlias)
RETURN

```


Este modelo de programa exibe um cabeçalho com informações pré-determinadas, um Browse horizontal central (dependente do dicionário de dados) e um rodapé com variáveis de memória que são atualizadas de acordo com os valores preenchidos no Browse horizontal.

As validações do cabeçalho são pré-determinadas no programa-fonte. Já as validações do browse horizontal são genericamente herdadas do dicionário de dados.



```

/* /
+-----+
+ Funcao   | CTBA120 | Autor   | Pilar S. Albaladejo | Data   | 24/07/00 |
+-----+-----+-----+-----+-----+-----+
| Descricao | Cadastro de Criterios de Rateio Externo
+-----+-----+-----+-----+-----+-----+
| Sintaxe   | CTBA120()
+-----+-----+-----+-----+-----+-----+
| Uso       | Generico
+-----+-----+-----+-----+-----+-----+
|           | ATUALIZACOES SOFRIDAS DESDE A CONSTRUCAO NICIAL
+-----+-----+-----+-----+-----+-----+
| Programador | Data   | BOPS   | Motivo da Alteracao
+-----+-----+-----+-----+-----+-----+
|           |       |       |
+-----+-----+-----+-----+-----+-----+
/* /
#include "CTBA120.CH"
#include "PROTHEUS.CH"
#include "FONT.CH"

```

```

FUNCTION CTBA120()
/*
+-----+
| Define Array contendo as Rotinas a executar do programa      +
| ----- Elementos contidos por dimensao -----              +
| 1. Nome a aparecer no cabecalho                               +
| 2. Nome da Rotina associada                                    +
| 3. Usado pela rotina                                          +
| 4. Tipo de Transacao a ser efetuada                           +
|   1 - Pesquisa e Posiciona em um Banco de Dados              +
|   2 - Simplesmente Mostra os Campos                          +
|   3 - Inclui registros no Bancos de Dados                    +
|   4 - Altera o registro corrente                             +
|   5 - Remove o registro corrente do Banco de Dados           +
+-----+
*/
PRIVATE aRotina := { { OemToAnsi(STR0001),"AxPesqui", 0 , 1},; // "Pesquisar"
                    { OemToAnsi(STR0002),"Ctbl20Cad", 0 , 2},; // "Visualizar"
                    { OemToAnsi(STR0003),"Ctbl20Cad", 0 , 3},; // "Incluir"
                    { OemToAnsi(STR0004),"Ctbl20Cad", 0 , 4},; // "Alterar"
                    { OemToAnsi(STR0005),"Ctbl20Cad", 0 , 5} } // "Excluir"

+-----+
| Define o cabecalho da tela de atualizacoes                    |
+-----+
Private cCadastro := OemToAnsi(STR0006) // "Critérios de Rateio"
+-----+
| Endereca funcao Mbrowse                                       |
+-----+
mBrowse( 6, 1,22,75,"CTJ" )
Return
/*
+-----+-----+-----+-----+-----+-----+
| Funcao      |CTB120CAD| Autor | Pilar S. Albaladejo | Data | 24/07/00 |
+-----+-----+-----+-----+-----+-----+
| Descricao   | Cadastro de Rateio Externo |
+-----+-----+-----+-----+-----+
| Sintaxe     | Ctbl20Cad(ExpC1,ExpN1,ExpN2) |
+-----+-----+-----+-----+-----+
| Parametros  | ExpC1 = Alias do arquivo      |
|              | ExpN1 = Numero do registro    |
|              | ExpN2 = Numero da opcao selecionada |
+-----+-----+-----+-----+-----+
| Uso         | CTBA120                       |
+-----+-----+-----+-----+-----+
*/
Function Ctbl20Cad(cAlias,nReg,nOpc)
Local aSaveArea := GetArea()
Local aCampos := {}
Local aAlterar := {}
Local aTpSald := CTBCBOX("CTJ_TPSALD")
Local cArg
Local cRateio
Local cDescRat
LOCAL cMoedaLc
Local cTpSald
Local nOpc := 0
Local oGetDb
Local oDlg
Local oFnt
Local oTpSald
Private aTela := {}
Private aGets := {}
Private aHeader := {}
Private nTotalD := 0
Private nTotalC := 0
+-----+
| Monta aHeader para uso com MSGETDB                             |
+-----+
aCampos := Ctbl20Head(@aAlterar)
+-----+
| Cria arquivo Temporario para uso com MSGETDB                     |

```

```

+-----+
Ctbl20Cri(aCampos,@cArq)
+-----+
| Carrega dados para MSGETDB |
+-----+
Ctbl20Carr(nOpc)
If nOpc == 3 // Inclusao
    cRateio := CriaVar("CTJ_RATEIO") // Numero do Rateio
    cDescRat := CriaVar("CTJ_DESC") // Descricao do Rateio
    cMoedaLC := CriaVar("CTJ_MOEDLC") // Moeda do Lancamento
    cTpSald := CriaVar("CTJ_TPSALD") // Tipo do Saldo
Else // Visualizacao / Alteracao / Exclusao
    cRateio := CTJ->CTJ_RATEIO
    cDescRat := CTJ->CTJ_DESC
    cMoedaLC := CTJ->CTJ_MOEDLC
    cTpSald := CTJ->CTJ_TPSALD
EndIf
+-----+
| Monta Tela Modelo 2 |
+-----+
DEFINE MSDIALOG oDlg TITLE OemToAnsi(STR0006) From 9,0 To 32,80 OF oMainWnd // "Rateios
Externos"
DEFINE FONT oFnt NAME "Arial" Size 10,15
@ 18, 007 SAY OemToAnsi(STR0007) PIXEL // "Rateio: "

@ 18, 037 MSGET cRateio Picture "9999" SIZE 020,08 When (nOpc == 3);
Valid Ctbl20Rat(cRateio) OF oDlg PIXEL

@ 18, 090 Say OemToAnsi(STR0008) PIXEL // "Descricao: "
@ 18, 120 MSGET cDescRat Picture "@" SIZE 140,08 When (nOpc == 3 .Or. ;
nOpc == 4) Valid !Empty(cDescRat) OF oDlg PIXEL
@ 33, 007 Say OemToAnsi(STR0009) PIXEL // "Moeda:"
@ 32, 037 MSGET cMoedaLc Picture "@" F3 "CTO" SIZE 020,08 When (nOpc == 3 .Or. ;
nOpc == 4) Valid Ctbl20Moed(cMoedaLC) Of oDlg PIXEL

@ 33, 090 SAY OemToAnsi(STR0010) PIXEL // "Saldo:"
@ 32, 120 MSCOMBOBOX oTpSald VAR cTpSald ITEMS aTpSald When (nOpc == 3 .Or. ;
nOpc == 4) SIZE 45,08 OF oDlg PIXEL Valid (!Empty(cTpSald) .And.;
CtblTpSald(@cTpSald,aTpSald))
+-----+
| Chamada da MSGETDB |
+-----+
oGetDB := MSGetDB():New(044, 005, 120, 315, If(nOpc==3,4,nOpc),"CTB120LOK",;
"CTB120TOK", "+CTJ_SEQUEN",.t.,aAlterar,.t.,,"TMP")
+-----+
| Validacao da janela |
+-----+
ACTIVATE MSDIALOG oDlg ON INIT EnchoiceBar(oDlg,;
{| |nOpca:=1.if(Ctbl20TOK(),oDlg:End(),nOpca := 0)},;
{| |nOpca:=2,oDlg:End()}) VALID nOpca != 0
IF nOpca == 1 // Aceita operacao e grava dados
    Begin Transaction
    Ctbl20Gra(cRateio,cDescRat,nOpc,cMoedaLC,cTpSald)
    End Transaction
ENDIF
dbSelectArea(cAlias)
+-----+
| Apaga arquivo temporario gerado para MSGETDB |
+-----+
DbSelectArea( "TMP" )
DbCloseArea()
If Select("cArq") = 0
    FErase(cArq+GetDBExtension())
EndIf
dbSelectArea("CTJ")
dbSetOrder(1)
Return nOpca
/*
+-----+-----+-----+-----+-----+-----+
| Funcao      |CTB120RAT| Autor | Pilar S. Albaladejo | Data | 24/07/00 |
+-----+-----+-----+-----+-----+-----+

```

Descricao	Verifica existencia do Rateio
Sintaxe	Ctbl20Rat(ExpC1)
Parametros	ExpC1 = Numero do Rateio
Retorno	.T./.F.
Uso	CTBA120

```

/*
Function Ctbl20Rat(cRateio)
Local aSaveArea:= GetArea()
Local lRet := .T.
Local nReg
If Empty(cRateio)
    lRet := .F.
Else
    dbSelectArea("CTJ")
    dbSetOrder(1)
    nReg := Recno()
    If dbSeek(xFilial()+cRateio)
        Help(" ",1,"CTJNRATEIO")
        lRet := .F.
    EndIf
    dbGoto(nReg)
EndIf
RestArea(aSaveArea)
Return lRet

/*


|            |                                                                                                                                                                 |       |                     |      |          |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|---------------------|------|----------|
| Funcao     | CTB120GRA                                                                                                                                                       | Autor | Pilar S. Albaladejo | Data | 24/07/00 |
| Descricao  | Grava resgistro digitados                                                                                                                                       |       |                     |      |          |
| Sintaxe    | Ctbl20Gra(ExpC1,ExpC2,ExpN1,cExpC3,cExpC4)                                                                                                                      |       |                     |      |          |
| Parametros | ExpC1 = Numero do Rateio<br>ExpC2 = Descricao do Rateio<br>ExpN1 = Opcao do Menu (Inclusao / Alteracao etc)<br>ExpC3 = Moeda do Rateio<br>ExpC4 = Tipo de Saldo |       |                     |      |          |
| Retorno    | Nenhum                                                                                                                                                          |       |                     |      |          |
| Uso        | CTBA120                                                                                                                                                         |       |                     |      |          |


Function Ctbl20Gra(cRateio,cDescRat,nOpc,cMoedaLC,cTpSald)
Local aSaveArea := GetArea()
dbSelectArea("TMP")
dbgotop()
While !Eof()
    If !TMP->CTJ_FLAG // Item nao deletado na MSGETDB
        If nOpc == 3 .Or. nOpc == 4
            dbSelectArea("CTJ")
            dbSetOrder(1)
            If !(dbSeek(xFilial()+cRateio+TMP->CTJ_SEQUEN))
                RecLock("CTJ",.t.)
                CTJ->CTJ_FILIAL := xFilial()
                CTJ->CTJ_RATEIO := cRateio
                CTJ->CTJ_DESC := cDescRat
                CTJ->CTJ_MOEDLC := cMoedaLC
                CTJ->CTJ_TPSALD := cTpSald
            Else
                RecLock("CTJ",.f.)
                CTJ->CTJ_DESC := cDescRat
                CTJ->CTJ_MOEDLC := cMoedaLC
                CTJ->CTJ_TPSALD := cTpSald
            Endif
            For nCont := 1 To Len(aHeader)

```

```

        If (aHeader[nCont][10] != "V" )
            FieldPut(FieldPos(aHeader[nCont][2]),;
                TMP->(FieldGet(FieldPos(aHeader[nCont][2]))))
        EndIf
        Next nCont
        MsUnlock()
    ElseIf nOpc == 5 // Se for exclusao
        dbSelectArea("CTJ")
        dbSetOrder(1)
        If dbSeek(xFilial()+cRateio+TMP->CTJ_SEQUEN)
            RecLock("CTJ",.F.,.T.)
            dbDelete()
            MsUnlock()
        EndIf
    EndIf
Else // Item deletado na MSGETDB
    dbSelectArea("CTJ")
    dbSetOrder(1)
    If dbSeek(xFilial()+cRateio+TMP->CTJ_SEQUEN)
        RecLock("CTJ",.f.,.t.)
        DbDelete()
        MsUnlock()
    Endif
EndIf
dbSelectArea("TMP")
dbSkip()
Enddo
RestArea(aSaveArea)
Return

/*
+-----+-----+-----+-----+-----+-----+
| Funcao   | CTB120TOK | Autor   | Pilar S. Albaladejo | Data   | 24/07/00 |
+-----+-----+-----+-----+-----+-----+
| Descricao | Valida MSGETDB -> Tudo OK
+-----+-----+-----+-----+-----+-----+
| Sintaxe   | Ctb120TOK(ExpC1)
+-----+-----+-----+-----+-----+-----+
| Parametros | Nenhum
+-----+-----+-----+-----+-----+-----+
| Retorno   | Nenhum
+-----+-----+-----+-----+-----+-----+
| Uso       | CTBA120
+-----+-----+-----+-----+-----+-----+
*/
Function Ctb120Tok()
Local aSaveArea := GetArea()
Local lRet := .T.
Local nTotalD := 0
Local nTotalC := 0
dbSelectArea("TMP")
dbGotop()
While !Eof()
    If !TMP->CTJ_FLAG
        If !Ctb120LOK()
            lRet := .F.
            Exit
        EndIf
        If !Empty(TMP->CTJ_DEBITO)
            nTotalD += TMP->CTJ_PERCEN
        EndIf
        If !Empty(TMP->CTJ_CREDITO)
            nTotalC += TMP->CTJ_PERCEN
        EndIf
    EndIf
    dbSkip()
Enddo
nTotalD := Round(nTotalD,2)
nTotalC := Round(nTotalC,2)
If lRet
    IF (nTotalD > 0 .And. nTotalD != 100 ).Or. (nTotalC > 0 .And. nTotalC != 100)

```

```

        Help(" ",1,"CTJ100%")
        lRet := .F.
    EndIf
EndIf
RestArea(aSaveArea)
Return lRet
*/
+-----+-----+-----+-----+-----+
| Funcao   |CTB120LOK| Autor   | Pilar S. Albaladejo   | Data   | 24/07/00 |
+-----+-----+-----+-----+-----+
| Descricao | Valida MSGETDB -> LinhaOK |
+-----+-----+-----+-----+
| Sintaxe   | Ctbl20LOK(ExpC1) |
+-----+-----+-----+-----+
| Parametros | Nenhum |
+-----+-----+-----+-----+
| Retorno   | Nenhum |
+-----+-----+-----+-----+
| Uso       | CTBA120 |
+-----+-----+-----+-----+
*/
Function CTB120LOK()
Local lRet := .T.
Local nCont
If !TMP->CTJ_FLAG
    If Empty(TMP->CTJ_PERCEN)
        Help(" ",1,"CTJVLZERO")
        lRet := .F.
    EndIf
    If lRet
        ValidaConta(TMP->CTJ_DEBITO,"1",,,.t.)
    EndIf
    If lRet
        ValidaConta(TMP->CTJ_CREDITO,"2",,,.T.)
    EndIf
EndIf
Return lRet
*/
+-----+-----+-----+-----+-----+
| Funcao   |CTB120Cri| Autor   | Pilar S. Albaladejo   | Data   | 24/07/00 |
+-----+-----+-----+-----+-----+
| Descricao | Cria Arquivo Temporario para MSGETDB |
+-----+-----+-----+-----+
| Sintaxe   | Ctbl20Cri(ExpA1,ExpC1) |
+-----+-----+-----+-----+
| Parametros | ExpA1 = Matriz com campos a serem criados |
|           | ExpC1 = Nome do arquivo temporario |
+-----+-----+-----+-----+
| Retorno   | Nenhum |
+-----+-----+-----+-----+
| Uso       | CTBA120 |
+-----+-----+-----+-----+
*/
Function Ctbl20Cria(aCampos,cArq)
Local cChave
Local aSaveArea := GetArea()
cChave := "CTJ_SEQUEN"
cArq := CriaTrab(aCampos,.t.)
dbUseArea(.t.,cArq,"TMP",.f.,.f.)
RestArea(aSaveArea)
Return
*/
+-----+-----+-----+-----+-----+
| Funcao   |CTB120Head| Autor   | Pilar S. Albaladejo   | Data   | 24/07/00 |
+-----+-----+-----+-----+-----+
| Descricao | Montar aHeader para arquivo temporario da MSGETDB |
+-----+-----+-----+-----+
| Sintaxe   | Ctbl20Head(ExpA1) |
+-----+-----+-----+-----+
| Parametros | ExpA1 = Matriz com campos que podem ser alterados |
+-----+-----+-----+-----+

```

```

| Retorno      | ExpA1 = Matriz com campos a serem criados no arq temporario|
+-----+-----+
| Uso          | CTBA120                                                    |
+-----+-----+
*/
Function Ctbl20Head(aAlterar)
Local aSaveArea:= GetArea()
Local aFora := {"CTJ_RATEIO","CTJ_DESC","CTJ_MOEDLC","CTJ_TPSALD","CTJ_VALOR"}
Local aCampos := {}
Local nCriter := 0
PRIVATE nUsado := 0
// Montagem da matriz aHeader
dbSelectArea("SX3")
dbSetOrder(1)
dbSeek("CTJ")
While !EOF().And. (x3_arquivo == "CTJ")
  If Alltrim(x3_campo) == "CTJ_SEQUEN" .Or. ;
  x3Uso(x3_usado) .and. cNivel >= x3_nivel
    If Ascan(aFora,Trim(X3_CAMPO)) <= 0
      nUsado++
      AADD(aHeader,{ TRIM(X3Titulo()), x3_campo, x3_picture,;
      x3_tamanho, x3_decimal, x3_valid,;
      x3_usado, x3_tipo, "TMP", x3_context } )
      If Alltrim(x3_campo) <> "CTJ_SEQUEN"
        Add(aAlterar,Trim(X3_CAMPO))
      EndIf
    EndIf
  EndIf
  aAdd( aCampos, { SX3->X3_CAMPO, SX3->X3_TIPO, SX3->X3_TAMANHO,;
  SX3->X3_DECIMAL } )
  dbSkip()
EndDO
Add(aCampos,{"CTJ_FLAG","L",1,0})
RestArea(aSaveArea)
Return aCampos
*/
+-----+-----+-----+-----+-----+-----+
| Funcao      |CTB120Carr| Autor | Pilar S. Albaladejo | Data | 24/07/00 |
+-----+-----+-----+-----+-----+-----+
| Descricao   | Carrega dados para MSGETDB |
+-----+-----+-----+-----+-----+-----+
| Sintaxe     | Ctbl20Carr(ExpN1) |
+-----+-----+-----+-----+-----+-----+
| Parametros  | ExpN1 = Opcao do Menu -> Inclusao / Alteracao etc |
+-----+-----+-----+-----+-----+-----+
| Retorno     | Nenhum |
+-----+-----+-----+-----+-----+-----+
| Uso        | CTBA120 |
+-----+-----+-----+-----+-----+-----+
*/
Function CTB120Carr(nOpc)
Local aSaveArea:= GetArea()
Local cAlias := "CTJ"
Local nPos
If nOpc != 3 // Visualizacao / Alteracao / Exclusao
  cRateio := CTJ->CTJ_RATEIO
  dbSelectArea("CTJ")
  dbSetOrder(1)
  If dbSeek(xFilial()+cRateio)
    While !Eof().And. CTJ->CTJ_FILIAL == xFilial().And.;
    CTJ->CTJ_RATEIO == cRateio
      dbSelectArea("TMP")
      dbAppend()
      For nCont := 1 To Len(aHeader)
        nPos := FieldPos(aHeader[nCont][2])
        If (aHeader[nCont][08] <> "M" .And. aHeader[nCont][10] <> "V" )
          FieldPut(nPos,(cAlias)-
          >(FieldGet(FieldPos(aHeader[nCont][2]))))
        EndIf
      Next nCont
      TMP->CTJ_FLAG := .F.
    EndWhile
  EndIf
EndIf

```

```


                dbSelectArea("CTJ")
                dbSkip()
            EndDo
        EndIf
    Else
        dbSelectArea("TMP")
        dbAppend()
        For nCont := 1 To Len(aHeader)
            If (aHeader[nCont][08] <> "M" .And. aHeader[nCont][10] <> "V" )
                nPos := FieldPos(aHeader[nCont][2])
                FieldPut(nPos,CriaVar(aHeader[nCont][2],.T.))
            EndIf
        Next nCont
        TMP->CTJ_FLAG := .F.
        TMP->CTJ_SEQUEN:= "001"
    EndIf
    dbSelectArea("TMP")
    dbGoTop()
    RestArea(aSaveArea)
    Return
    /*
+-----+-----+-----+-----+-----+-----+
| Funcao   |CT120Moed| Autor   | Pilar S. Albaladejo | Data   | 24/07/00 |
+-----+-----+-----+-----+-----+-----+
| Descricao | Valida Moeda do Lancamento
+-----+-----+-----+-----+-----+-----+
| Sintaxe   | Ctb120Moed(ExpC1)
+-----+-----+-----+-----+-----+-----+
| Parametros | ExpC1 = Moeda a ser validada
+-----+-----+-----+-----+-----+-----+
| Retorno   | .T./.F.
+-----+-----+-----+-----+-----+-----+
| Uso       | CTBA120
+-----+-----+-----+-----+-----+-----+
*/
Function Ctl120MoedLC(cMoeda)
Local aCtbMoeda:= {}
Local lRet := .T.
aCtbMoeda := CtbMoeda(cMoeda)
If Empty(aCtbMoeda[1])
    Help(" ",1,"NOMOEDA")
    lRet := .F.
Endif
Return lRet

```

Modelo 3

Este modelo de programa é a junção dos Modelos 1 e 2. Nele é exibido um Browse vertical e um Browse horizontal (ambos dependentes do dicionário de dados). As validações são genericamente herdadas do dicionário de dados.

AP6 Remote - Faturamento [Processo de Venda]



Processo de Venda - Altera

Processo: 000001 Descrição: ADVANCED CLASSIC

Observacoes:

Estágio	Descrição	Tarefas	Contribui %
000001	VISITA DE CONTATO	Memo	
000002	PRIMEIRA DEMO	Memo	
000003	SEGUNDA DEMO	Memo	
000004	CONTRATO	Memo	
000005	FECHAMENTO	Memo	

```

/* /
+-----+
+ Funcao   | FATA010 | Autor | Eduardo Riera      | Data | 11/01/00 |
+-----+-----+
+ Descricao | Cadastro de Processo de Vendas |
+-----+-----+
+ Sintaxe   | FATA010() |
+-----+-----+
+ Uso       | Generico  |
+-----+-----+
+          | ATUALIZACOES SOFRIDAS DESDE A CONSTRUCAO NICIAL |
+-----+-----+
+ Programador | Data   | BOPS | Motivo da Alteracao |
+-----+-----+
+          |       |     |                     |
+-----+-----+
/* /
#include "FATA010.CH"
#include "FIVEWIN.CH"
#define APOS { 15, 1, 70, 315 }
Function Fata010()
/* /
+-----+
+ Define Array contendo as Rotinas a executar do programa +
+ ----- Elementos contidos por dimensao ----- +
+ 1. Nome a aparecer no cabeçalho +
+ 2. Nome da Rotina associada +
+ 3. Usado pela rotina +

```

```

| 4. Tipo de Transacao a ser efetuada +
| 1 - Pesquisa e Posiciona em um Banco de Dados +
| 2 - Simplesmente Mostra os Campos +
| 3 - Inclui registros no Bancos de Dados +
| 4 - Altera o registro corrente +
| 5 - Remove o registro corrente do Banco de Dados +
+-----+
/*
PRIVATE cCadastro := OemToAnsi(STR0001) //"Processo de Venda"
PRIVATE aRotina := { { OemToAnsi(STR0002),"AxPesqui",0,1},; //"Pesquisar"
                    { OemToAnsi(STR0003),"Ft010Visua",0,2},; //"Visual"
                    { OemToAnsi(STR0004),"Ft010Inclu",0,3},; //"Incluir"
                    { OemToAnsi(STR0005),"Ft010Alter",0,4},; //"Alterar"
                    { OemToAnsi(STR0006),"Ft010Exclu",0,5} } //"Exclusao"

If !Empty( Select( "AC9" ) )
    AAdd( aRotina, { STR0013,"MsDocument",0,4} )
EndIf
mBrowse( 6, 1,22,75,"AC1" )
Return(.T.)

/*
+-----+-----+-----+-----+-----+-----+
| Funcao |Ft010Visua| Autor |Eduardo Riera | Data |13.01.2000|
+-----+-----+-----+-----+-----+-----+
| Descricao |Funcao de Tratamento da Visualizacao |
+-----+-----+-----+-----+-----+-----+
| Sintaxe | Ft010Visua(ExpC1,ExpN2,ExpN3) |
+-----+-----+-----+-----+-----+-----+
| Parametros | ExpC1: Alias do arquivo |
| | ExpN2: Registro do Arquivo |
| | ExpN3: Opcao da MBrowse |
+-----+-----+-----+-----+-----+-----+
| Retorno | Nenhum |
+-----+-----+-----+-----+-----+-----+
| Uso | FATA010 |
+-----+-----+-----+-----+-----+-----+
/*
Function Ft010Visua(cAlias,nReg,nOpcx)
Local aArea := GetArea()
Local oGetDad
Local oDlg
Local nUsado := 0
Local nCntFor := 0
Local nOpcA := 0
Local lContinua := .T.
Local lQuery := .F.
Local cCadastro := OemToAnsi(STR0001) //"Processo de Venda"
Local cQuery := ""
Local cTrab := "AC2"
Local bWhile := {|| .T. }
Local aObjects := {}
Local aPosObj := {}
Local aSizeAut := MsAdvSize()
PRIVATE aHEADER := {}
PRIVATE aCOLS := {}
PRIVATE aGETS := {}
PRIVATE aTELA := {}

+-----+-----+
| Montagem de Variaveis de Memoria |
+-----+-----+
dbSelectArea( "AC1" )
dbSetOrder(1)
For nCntFor := 1 To FCount()
    M->&(FieldName(nCntFor)) := FieldGet(nCntFor)
Next nCntFor
+-----+-----+
| Montagem do aHeader |
+-----+-----+
dbSelectArea( "SX3" )
dbSetOrder(1)

```

```

dbSeek("AC2")
While ( !Eof() .And. SX3->X3_ARQUIVO == "AC2" )
  If ( X3USO(SX3->X3_USADO) .And. cNivel >= SX3->X3_NIVEL )
    nUsado++
    Aadd(aHeader,{ TRIM(X3Titulo()),;
    TRIM(SX3->X3_CAMPO),;
    SX3->X3_PICTURE,;
    SX3->X3_TAMANHO,;
    SX3->X3_DECIMAL,;
    SX3->X3_VALID,;
    SX3->X3_USADO,;
    SX3->X3_TIPO,;
    SX3->X3_ARQUIVO,;
    SX3->X3_CONTEXT } )
  EndIf
  dbSelectArea("SX3")
  dbSkip()
EndDo
+-----+
|  Montagem do aCols  |
+-----+
dbSelectArea("AC2")
dbSetOrder(1)
#IFDEF TOP
If ( TcSrvType()!="AS/400" )
  lQuery := .T.
  cQuery := "SELECT *,R_E_C_N_O_ AC2RECNO "
  cQuery += "FROM "+RetSqlName("AC2")+" AC2 "
  cQuery += "WHERE AC2.AC2_FILIAL='"+xFilial("AC2")+"' AND "
  cQuery += "AC2.AC2_PROVEN='"+AC1->AC1_PROVEN+"' AND "
  cQuery += "AC2.D_E_L_E_T_<>'*'"
  cQuery += "ORDER BY "+SqlOrder(AC2->(IndexKey()))

  cQuery := ChangeQuery(cQuery)
  cTrab := "FT010VIS"
  dbUseArea(.T., "TOPCONN", TcGenQry(, cQuery), cTrab, .T., .T.)
  For nCntFor := 1 To Len(aHeader)
    TcSetField(cTrab, AllTrim(aHeader[nCntFor][2]), aHeader[nCntFor,8], aHeader[nCnt
    For,4], aHeader[nCntFor,5])
  Next nCntFor
Else
#ENDIF
  AC2->(dbSeek(xFilial("AC2")+AC1->AC1_PROVEN))
  bWhile := {|| xFilial("AC2") == AC2->AC2_FILIAL .And.;
  AC1->AC1_PROVEN == AC2->AC2_PROVEN }
#IFDEF TOP
EndIf
#ENDIF

While ( !Eof() .And. Eval(bWhile) )
  aadd(aCOLS, Array(nUsado+1))
  For nCntFor := 1 To nUsado
    If ( aHeader[nCntFor][10] != "V" )
      aCols[Len(aCols)][nCntFor] := FieldGet(FieldPos(aHeader[nCntFor][2]))
    Else
      If ( lQuery )
        AC2->(dbGoto((cTrab)->AC2RECNO))
      EndIf
      aCols[Len(aCols)][nCntFor] := CriaVar(aHeader[nCntFor][2])
    EndIf
  Next nCntFor
  aCOLS[Len(aCols)][Len(aHeader)+1] := .F.
  dbSelectArea(cTrab)
  dbSkip()
EndDo
If ( lQuery )
  dbSelectArea(cTrab)
  dbCloseArea()
  dbSelectArea(cAlias)
EndIf
aObjects := {}

```

```

AAdd( aObjects, { 315, 50, .T., .T. } )
AAdd( aObjects, { 100, 100, .T., .T. } )
aInfo := { aSizeAut[ 1 ], aSizeAut[ 2 ], aSizeAut[ 3 ], aSizeAut[ 4 ], 3, 3 }
aPosObj := MsObjSize( aInfo, aObjects, .T. )
DEFINE MSDIALOG oDlg TITLE cCadastro From aSizeAut[7],00 To aSizeAut[6],aSizeAut[5] OF
oMainWnd PIXEL
EnChoice( cAlias ,nReg, nOpcx, , , , aPosObj[1], , 3 )
oGetDad := MSGetDados():New (aPosObj[2,1], aPosObj[2,2], aPosObj[2,3], aPosObj[2,4],
nOpcx, "Ft010LinOk", "AlwaysTrue", "",.F.)
ACTIVATE MSDIALOG oDlg ON INIT EnchoiceBar(oDlg,{||oDlg:End()},{||oDlg:End()})
RestArea(aArea)
Return(.T.)
*/
+-----+-----+-----+-----+-----+-----+
| Funcao      |Ft010Inclu|Autor |Eduardo Riera      | Data |13.01.2000|
|-----+-----+-----+-----+-----+-----+
| Descricao   |Funcao de Tratamento da Inclusao
+-----+-----+-----+-----+-----+
| Sintaxe     | Ft010Inclu(ExpC1,ExpN2,ExpN3)
+-----+-----+-----+-----+-----+
| Parametros  | ExpC1: Alias do arquivo
|              | ExpN2: Registro do Arquivo
|              | ExpN3: Opcao da MBrowse
+-----+-----+-----+-----+-----+
| Retorno     | Nenhum
+-----+-----+-----+-----+-----+
| Uso         | FATA010
+-----+-----+-----+-----+-----+
*/
Function Ft010Inclu(cAlias,nReg,nOpcx)
Local aArea      := GetArea()
Local cCadastro  := OemToAnsi(STR0001) //"Processo de Venda"
Local oGetDad
Local oDlg
Local nUsado      := 0
Local nCntFor     := 0
Local nOpcA      := 0
Local aObjects    := {}
Local aPosObj     := {}
Local aSizeAut    := MsAdvSize()
PRIVATE aHEADER   := {}
PRIVATE aCOLS     := {}
PRIVATE aGETS     := {}
PRIVATE aTELA     := {}
+-----+-----+-----+-----+-----+
| Montagem das Variaveis de Memoria
+-----+-----+-----+-----+-----+
dbSelectArea("AC1")
dbSetOrder(1)
For nCntFor := 1 To FCount()
    M->(FieldName(nCntFor)) := CriaVar(FieldName(nCntFor))
Next nCntFor
+-----+-----+-----+-----+-----+
| Montagem da aHeader
+-----+-----+-----+-----+-----+
dbSelectArea("SX3")
dbSetOrder(1)
dbSeek("AC2")
While ( !Eof() .And. SX3->X3_ARQUIVO == "AC2" )
    If ( X3USO(SX3->X3_USADO) .And. cNivel >= SX3->X3_NIVEL )
        nUsado++
        Aadd(aHeader,{ TRIM(X3Titulo()),;
            TRIM(SX3->X3_CAMPO),;
            SX3->X3_PICTURE,;
            SX3->X3_TAMANHO,;
            SX3->X3_DECIMAL,;
            SX3->X3_VALID,;
            SX3->X3_USADO,;
            SX3->X3_TIPO,;
            SX3->X3_ARQUIVO,;
            SX3->X3_CONTEXT } )
    EndIf
EndWhile

```

```

EndIf
dbSelectArea("SX3")
dbSkip()
EndDo
+-----+
| Montagem da Acols |
+-----+
aadd(aCOLS,Array(nUsado+1))
For nCntFor := 1 To nUsado
    aCols[1][nCntFor] := CriaVar(aHeader[nCntFor][2])
Next nCntFor
aCOLS[1][Len(aHeader)+1] := .F.
aObjects := {}
AAdd( aObjects, { 315, 50, .T., .T. } )
AAdd( aObjects, { 100, 100, .T., .T. } )
aInfo := { aSizeAut[ 1 ], aSizeAut[ 2 ], aSizeAut[ 3 ], aSizeAut[ 4 ], 3, 3 }
aPosObj := MSObjSize( aInfo, aObjects, .T. )
DEFINE MSDIALOG oDlg TITLE cCadastro From aSizeAut[7],00 To aSizeAut[6],aSizeAut[5] OF
oMainWnd PIXEL
EnChoice( cAlias ,nReg, nOpcx, , , , , aPosObj[1], , 3 )
oGetDad := MSGetDados():New(aPosObj[2,1], aPosObj[2,2], aPosObj[2,3], aPosObj[2,4],
nOpcx, "Ft010LinOk", "Ft010TudOk","",.T.)
ACTIVATE MSDIALOG oDlg ;
ON INIT EnChoiceBar(oDlg, {|nOpcA:=If(oGetDad:TudoOk() .And. Obrigatorio(aGets,aTela),
1,0),If(nOpcA=1,oDlg:End(),Nil)},{|oDlg:End()})
If ( nOpcA == 1 )
    Begin Transaction
    Ft010Grv(1)
    If ( __lSX8 )
        ConfirmSX8()
    EndIf
    EvalTrigger()
    End Transaction
Else
    If ( __lSX8 )
        RollBackSX8()
    EndIf
EndIf
EndIf
RestArea(aArea)
Return(.T.)
*/
+-----+-----+-----+-----+-----+-----+
| Funcao |Ft010Alter| Autor |Eduardo Riera | Data |13.01.2000|
+-----+-----+-----+-----+-----+-----+
| Descricao |Funcao de Tratamento da Alteracao |
+-----+-----+-----+-----+-----+-----+
| Sintaxe | Ft010Alter(ExpC1,ExpN2,ExpN3) |
+-----+-----+-----+-----+-----+-----+
| Parametros | ExpC1: Alias do arquivo |
| | ExpN2: Registro do Arquivo |
| | ExpN3: Opcao da MBrowse |
+-----+-----+-----+-----+-----+-----+
| Retorno | Nenhum |
+-----+-----+-----+-----+-----+-----+
| Uso | FATA010 |
+-----+-----+-----+-----+-----+-----+
*/
Function Ft010Alter(cAlias,nReg,nOpcx)
Local aArea := GetArea()
Local cCadastro := OemToAnsi(STR0001) //"Processo de Venda"
Local oGetDad
Local oDlg
Local nUsado := 0
Local nCntFor := 0
Local nOpcA := 0
Local lContinua := .T.
Local cQuery := ""
Local cTrab := "AC2"
Local bWhile := {|.T. }
Local aObjects := {}
Local aPosObj := {}

```

```

Local aSizeAut := MsAdvSize()
PRIVATE aHEADER := {}
PRIVATE aCOLS := {}
PRIVATE aGETS := {}
PRIVATE aTELA := {}
+-----+
|  Montagem das Variaveis de Memoria  |
+-----+
dbSelectArea("AC1")
dbSetOrder(1)
lContinua := SoftLock("AC1")
If ( lContinua )
    For nCntFor := 1 To FCount()
        M->&(FieldName(nCntFor)) := FieldGet(nCntFor)
    Next nCntFor
+-----+
|  Montagem da aHeader  |
+-----+
dbSelectArea("SX3")
dbSetOrder(1)
dbSeek("AC2")
While ( !Eof() .And. SX3->X3_ARQUIVO == "AC2" )
    If ( X3USO(SX3->X3_USADO) .And. cNivel >= SX3->X3_NIVEL )
        nUsado++
        Add(aHeader,{ TRIM(X3Titulo()),;
            TRIM(SX3->X3_CAMPO),;
            SX3->X3_PICTURE,;
            SX3->X3_TAMANHO,;
            SX3->X3_DECIMAL,;
            SX3->X3_VALID,;
            SX3->X3_USADO,;
            SX3->X3_TIPO,;
            SX3->X3_ARQUIVO,;
            SX3->X3_CONTEXT } )
    EndIf
    dbSelectArea("SX3")
    dbSkip()
EndDo
+-----+
|  Montagem da aCols  |
+-----+
dbSelectArea("AC2")
dbSetOrder(1)
#IFDEF TOP
If ( TcSrvType()!="AS/400" )
    lQuery := .T.
    cQuery := "SELECT *,R_E_C_N_O_ AC2RECNO "
    cQuery += "FROM "+RetSqlName("AC2")+" AC2 "
    cQuery += "WHERE AC2.AC2_FILIAL='"+xFilial("AC2")+"' AND "
    cQuery += "AC2.AC2_PROVEN='"+AC1->AC1_PROVEN+"' AND "
    cQuery += "AC2.D_E_L_E_T_<>'*'"
    cQuery += "ORDER BY "+SqlOrder(AC2->(IndexKey()))

    cQuery := ChangeQuery(cQuery)
    cTrab := "FT010VIS"
    dbUseArea(.T., "TOPCONN", TcGenQry(, , cQuery), cTrab, .T., .T.)
    For nCntFor := 1 To Len(aHeader)
        TcSetField(cTrab, AllTrim(aHeader[nCntFor][2]), aHeader[nCntFor, 8], ;
            Header[nCntFor, 4], aHeader[nCntFor, 5])
    Next nCntFor
Else
#ENDIF
    AC2->(dbSeek(xFilial("AC2")+AC1->AC1_PROVEN))
    bWhile := {| xFilial("AC2") == AC2->AC2_FILIAL .And.;
        AC1->AC1_PROVEN == AC2->AC2_PROVEN }
#IFDEF TOP
EndIf
#ENDIF
While ( !Eof() .And. Eval(bWhile) )
    aadd(aCOLS, Array(nUsado+1))
    For nCntFor := 1 To nUsado

```

```

        If ( aHeader[nCntFor][10] != "V" )
            aCols[Len(aCols)][nCntFor] :=
                FieldGet(FieldPos(aHeader[nCntFor][2]))
        Else
            If ( lQuery )
                AC2->(dbGoto((cTrab)->AC2RECNO))
            EndIf
            aCols[Len(aCols)][nCntFor] := CriaVar(aHeader[nCntFor][2])
        EndIf
    Next nCntFor
    aCOLS[Len(aCols)][Len(aHeader)+1] := .F.
    dbSelectArea(cTrab)
    dbSkip()
EndDo
If ( lQuery )
    dbSelectArea(cTrab)
    dbCloseArea()
    dbSelectArea(cAlias)
EndIf
EndIf
If ( lContinua )
    aObjects := {}
    AAdd( aObjects, { 315, 50, .T., .T. } )
    AAdd( aObjects, { 100, 100, .T., .T. } )
    aInfo := { aSizeAut[ 1 ], aSizeAut[ 2 ], aSizeAut[ 3 ], aSizeAut[ 4 ], 3, 3 }
    aPosObj := MsObjSize( aInfo, aObjects, .T. )
    DEFINE MSDIALOG oDlg TITLE cCadastro From aSizeAut[7],00 To aSizeAut[6],aSizeAut[5]
    ;
    OF MainWnd PIXEL
    EnChoice( cAlias ,nReg, nOpcx, , , , aPosObj[1], , 3 )
    oGetDad :=
        MSGetDados():New(aPosObj[2,1],aPosObj[2,2],aPosObj[2,3],aPosObj[2,4],nOpcx,;
            "Ft010LinOk","Ft010TudOk","",.T.)
    ACTIVATE MSDIALOG oDlg ;
    ON INIT
    EnchoiceBar(oDlg,{||nOpca:=If(oGetDad:TudoOk().And.Obrigatorio(aGets,aTela),1,0),;
        If(nOpca=1,oDlg:End(),Nil)},{||oDlg:End()})
        If ( nOpca == 1 )
            Begin Transaction
            Ft010Grv(2)
            If ( __lSX8 )
                ConfirmSX8()
            EndIf
            EvalTrigger()
            End Transaction
        Else
            If ( __lSX8 )
                RollBackSX8()
            EndIf
        EndIf
    EndIf
EndIf
Endif
RestArea(aArea)
Return(.T.)
/*
+-----+-----+-----+-----+-----+-----+
| Funcao   | Ft010Exclu | Autor | Eduardo Riera          | Data | 13.01.2000 |
+-----+-----+-----+-----+-----+-----+
| Descricao | Funcao de Tratamento da Exclusao |
+-----+-----+-----+-----+-----+-----+
| Sintaxe   | Ft010Exclu(ExpC1,ExpN2,ExpN3) |
+-----+-----+-----+-----+-----+-----+
| Parametros | ExpC1: Alias do arquivo |
|            | ExpN2: Registro do Arquivo |
|            | ExpN3: Opcao da MBrowse |
+-----+-----+-----+-----+-----+-----+
| Retorno   | Nenhum |
+-----+-----+-----+-----+-----+-----+
| Uso       | FATA010 |
+-----+-----+-----+-----+-----+-----+
*/

```

```

Function Ft010Exclu(cAlias,nReg,nOpcx)
Local aArea      := GetArea()
Local cCadastro  := OemToAnsi(STR0001) //"Processo de Venda"
Local oGetDad
Local oDlg
Local nUsado      := 0
Local nCntFor     := 0
Local nOpcA       := 0
Local lContinua  := .T.
Local cQuery      := ""
Local cTrab       := "AC2"
Local bWhile      := {||.T. }
Local aObjects    := {}
Local aPosObj     := {}
Local aSizeAut    := MsAdvSize()
PRIVATE aHEADER   := {}
PRIVATE aCOLS     := {}
PRIVATE aGETS     := {}
PRIVATE aTELA     := {}
+-----+
|  Montagem das Variaveis de Memoria  |
+-----+
dbSelectArea("AC1")
dbSetOrder(1)
lContinua := SoftLock("AC1")
If ( lContinua )
    For nCntFor := 1 To FCount()
        M->&(FieldName(nCntFor)) := FieldGet(nCntFor)
    Next nCntFor
+-----+
|  Montagem da aHeader  |
+-----+
dbSelectArea("SX3")
dbSetOrder(1)
dbSeek("AC2")
While ( !Eof() .And. SX3->X3_ARQUIVO == "AC2" )
    If ( X3USO(SX3->X3_USADO) .And. cNivel >= SX3->X3_NIVEL )
        nUsado++
        Aadd(aHeader,{ TRIM(X3Titulo()),;
            TRIM(SX3->X3_CAMPO),;
            SX3->X3_PICTURE,;
            SX3->X3_TAMANHO,;
            SX3->X3_DECIMAL,;
            SX3->X3_VALID,;
            SX3->X3_USADO,;
            SX3->X3_TIPO,;
            SX3->X3_ARQUIVO,;
            SX3->X3_CONTEXT } )
    EndIf
    dbSelectArea("SX3")
    dbSkip()
EndDo
+-----+
|  Montagem da aCols  |
+-----+
dbSelectArea("AC2")
dbSetOrder(1)
#IFDEF TOP
If ( TcSrvType()!="AS/400" )
    lQuery := .T.
    cQuery := "SELECT *,R_E_C_N_O_ AC2RECNO "
    cQuery += "FROM "+RetSqlName("AC2")+" AC2 "
    cQuery += "WHERE AC2.AC2_FILIAL='"+xFilial("AC2")+"' AND "
    cQuery += "AC2.AC2_PROVEN='"+AC1->AC1_PROVEN+"' AND "
    cQuery += "AC2.D_E_L_E_T_<>'*'"
    cQuery += "ORDER BY "+SqlOrder(AC2->(IndexKey()))

    cQuery := ChangeQuery(cQuery)
    cTrab := "FT010VIS"
    dbUseArea(.T.,"TOPCONN",TcGenQry(,cQuery),cTrab,.T.,.T.)
    For nCntFor := 1 To Len(aHeader)

```



```

        TcSetField(cTrab, AllTrim(aHeader[nCntFor][2]), aHeader[nCntFor, 8], ;
        aHeader[nCntFor, 4], aHeader[nCntFor, 5])
    Next nCntFor
Else
#ENDIF
    AC2->(dbSeek(xFilial("AC2")+AC1->AC1_PROVEN))
    bWhile := {|| xFilial("AC2") == AC2->AC2_FILIAL .And. ;
    AC1->AC1_PROVEN == AC2->AC2_PROVEN }
#IFDEF TOP
EndIf
#ENDIF
While ( !Eof() .And. Eval(bWhile) )
    aadd(aCOLS, Array(nUsado+1))
    For nCntFor := 1 To nUsado
        If ( aHeader[nCntFor][10] != "V" )
            aCols[Len(aCols)][nCntFor] :=
                FieldGet(FieldPos(aHeader[nCntFor][2]))
        Else
            If ( lQuery )
                AC2->(dbGoto((cTrab)->AC2RECNO))
            EndIf
            aCols[Len(aCols)][nCntFor] := CriaVar(aHeader[nCntFor][2])
        EndIf
    Next nCntFor
    aCOLS[Len(aCols)][Len(aHeader)+1] := .F.
    dbSelectArea(cTrab)
    dbSkip()
EndDo
If ( lQuery )
    dbSelectArea(cTrab)
    dbCloseArea()
    dbSelectArea(cAlias)
EndIf
EndIf
If ( lContinua )
    aObjects := {}
    AAdd( aObjects, { 315, 50, .T., .T. } )
    AAdd( aObjects, { 100, 100, .T., .T. } )
    aInfo := { aSizeAut[ 1 ], aSizeAut[ 2 ], aSizeAut[ 3 ], aSizeAut[ 4 ], 3, 3 }
    aPosObj := MsObjSize( aInfo, aObjects, .T. )

    DEFINE MSDIALOG oDlg TITLE cCadastro From aSizeAut[7], 00 To ;
    aSizeAut[6], aSizeAut[5] OF oMainWnd PIXEL

    EnChoice( cAlias, nReg, nOpcx, , , , aPosObj[1], , 3 )
    oGetDad :=
    MSGetDados():New(aPosObj[2,1], aPosObj[2,2], aPosObj[2,3], aPosObj[2,4], nOpcx, ;
    "Ft010LinOk", "Ft010TudOk", "", .F.)
    ACTIVATE MSDIALOG oDlg ;
    ON INIT
    EnchoiceBar(oDlg, {|| nOpcA:=If(oGetDad:TudoOk(), 1, 0), If(nOpcA==1, oDlg:End(), Nil)}, ;
    {|| oDlg:End()})
    If ( nOpcA == 1 )
        Begin Transaction
        If Ft010DelOk()
            Ft010Grv(3)
            EvalTrigger()
        EndIf
        End Transaction
    EndIf
EndIf
RestArea(aArea)
Return(.T.)

/*
+-----+-----+-----+-----+-----+-----+
| Funcao   |Ft010LinOk| Autor   |Eduardo Riera          | Data |13.01.2000|
+-----+-----+-----+-----+-----+-----+
| Descricao |Funcao de Validacao da linha OK
+-----+-----+-----+-----+-----+-----+
| Sintaxe   | Ft010LinOk()
+-----+-----+-----+-----+-----+-----+

```

```

+-----+-----+-----+-----+-----+-----+
| Parametros | Nenhum                                     |
+-----+-----+-----+-----+-----+-----+
| Retorno    | Nenhum                                     |
+-----+-----+-----+-----+-----+-----+
| Uso        | FATA010                                   |
+-----+-----+-----+-----+-----+-----+
/*
Function Ft010LinOk()
Local lRetorno:= .T.
Local nPStage := aScan(aHeader,{|x| AllTrim(x[2])=="AC2_STAGE"})
Local nPDescri:= aScan(aHeader,{|x| AllTrim(x[2])=="AC2_DESCRI"})
Local nCntFor := 0
Local nUsado := Len(aHeader)
If ( !aCols[n][nUsado+1] )
+-----+-----+-----+-----+-----+-----+
| Verifica os campos obrigatorios                                     |
+-----+-----+-----+-----+-----+-----+
If ( nPStage == 0 .Or. nPDescri == 0 )
    Help(" ",1,"OBRIGAT")
    lRetorno := .F.
EndIf
If ( lRetorno .And. (Empty(aCols[n][nPStage]) .Or. Empty(aCols[n][nPDescri])) )
    Help(" ",1,"OBRIGAT")
    lRetorno := .F.
EndIf
+-----+-----+-----+-----+-----+-----+
| Verifica se não há estagios repetidos                               |
+-----+-----+-----+-----+-----+-----+
If ( nPStage != 0 .And. lRetorno )
    For nCntFor := 1 To Len(aCols)
        If ( nCntFor != n .And. !aCols[nCntFor][nUsado+1] )
            If ( aCols[n][nPStage] == aCols[nCntFor][nPStage] )
                Help(" ",1,"FT010LOK01")
                lRetorno := .F.
            EndIf
        EndIf
    Next nCntFor
EndIf
Return(lRetorno)
*/
+-----+-----+-----+-----+-----+-----+
| Funcao      |Ft010Grv | Autor |Eduardo Riera          | Data |13.01.2000|
+-----+-----+-----+-----+-----+-----+
| Descricao   |Funcao de Gravacao do Processe de Venda
+-----+-----+-----+-----+-----+-----+
| Sintaxe     | Ft010Grv(ExpN1)
+-----+-----+-----+-----+-----+-----+
| Parametros  | ExpN1: Opcao do Menu (Inclusao / Alteracao / Exclusao)
+-----+-----+-----+-----+-----+-----+
| Retorno     | .T.
+-----+-----+-----+-----+-----+-----+
| Uso         | FATA010
+-----+-----+-----+-----+-----+-----+
/*
Static Function Ft010Grv(nOpc)
Local aArea := GetArea()
Local aUsrMemo := If( ExistBlock( "FT010MEM" ), ExecBlock( "FT010MEM", .F.,.F. ), {} )
Local aMemoAC1 := {}
Local aMemoAC2 := {}
Local aRegistro := {}
Local cQuery := ""
Local lGravou := .F.
Local nCntFor := 0
Local nCntFor2 := 0
Local nUsado := Len(aHeader)
Local nPStage := aScan(aHeader,{|x| AllTrim(x[2])=="AC2_STAGE"})
Local nPMEMO := aScan(aHeader,{|x| AllTrim(x[2])=="AC2_MEMO"})

If ValType( aUsrMemo ) == "A" .And. Len( aUsrMemo ) > 0

```

```

For nLoop := 1 to Len( aUsrMemo )
    If aUsrMemo[ nLoop, 1 ] == "AC1"
        AAdd( aMemoAC1, { aUsrMemo[ nLoop, 2 ], aUsrMemo[ nLoop, 3 ] } )

    ElseIf aUsrMemo[ nLoop, 1 ] == "AC2"
        AAdd( aMemoAC2, { aUsrMemo[ nLoop, 2 ], aUsrMemo[ nLoop, 3 ] } )

    EndIf
Next nLoop
EndIf
+-----+
| Guarda os registros em um array para atualizacao |
+-----+
dbSelectArea( "AC2" )
dbSetOrder( 1 )
#IFDEF TOP
If ( TcSrvType() != "AS/400" )
    cQuery := "SELECT AC2.R_E_C_N_O_ AC2RECNO "
    cQuery += "FROM "+RetSqlName( "AC2" )+" AC2 "
    cQuery += "WHERE AC2.AC2_FILIAL='"+xFilial( "AC2" )+"' AND "
    cQuery += "AC2.AC2_PROVEN='"+M->AC1_PROVEN+"' AND "
    cQuery += "AC2.D_E_L_E_T_<>'*'"
    cQuery += "ORDER BY "+SqlOrder( AC2->( IndexKey() ) )

    cQuery := ChangeQuery( cQuery )
    dbUseArea( .T., "TOPCONN", TcGenQry( , cQuery ), "FT010GRV", .T., .T. )
    dbSelectArea( "FT010GRV" )
    While ( !Eof() )
        aadd( aRegistro, AC2RECNO )
        dbSelectArea( "FT010GRV" )
        dbSkip()
    EndDo
    dbSelectArea( "FT010GRV" )
    dbCloseArea()
    dbSelectArea( "AC2" )
Else
#ENDIF
dbSeek( xFilial( "AC2" ) + M->AC1_PROVEN )
While ( !Eof() .And. xFilial( "AC2" ) == AC2->AC2_FILIAL .And. ;
    M->AC1_PROVEN == AC2->AC2_PROVEN )
    aadd( aRegistro, AC2->( RecNo() ) )
    dbSelectArea( "AC2" )
    dbSkip()
EndDo
#IFDEF TOP
EndIf
#ENDIF
Do Case
+-----+
| Inclusao / Alteracao |
+-----+
Case nOpc != 3
    For nCntFor := 1 To Len( aCols )
        If ( nCntFor > Len( aRegistro ) )
            If ( !aCols[ nCntFor ][ nUsado+1 ] )
                RecLock( "AC2", .T. )
            EndIf
        Else
            AC2->( dbGoto( aRegistro[ nCntFor ] ) )
            RecLock( "AC2" )
        EndIf
        If ( !aCols[ nCntFor ][ nUsado+1 ] )
            lGravou := .T.
            For nCntFor2 := 1 To nUsado
                If ( aHeader[ nCntFor2 ][ 10 ] != "V" )
                    FieldPut( FieldPos( aHeader[ nCntFor2 ][ 2 ] ), aCols[ nCntFor ][ nCntFor2 ] )
                EndIf
            Next nCntFor2
+-----+
| Grava os campos obrigatorios |

```

```

+-----+
AC2->AC2_FILIAL := xFilial("AC2")
AC2->AC2_PROVEN := M->AC1_PROVEN
If ( nPMemo != 0 .And. !Empty(aCols[nCntFor][nPMemo]))
    MSMM(AC2-
        >AC2_CODMEM,,,aCols[nCntFor][nPMemo],1,,,"AC2","AC2_CODMEM")
EndIf
+-----+
| Grava os campos memo de usuario |
+-----+
For nLoop := 1 To Len( aMemoAC2 )
    MSMM(AC2->(FieldGet(aMemoAC2[nLoop,1])),,, ;
        DFieldGet( aMemoAC2[nLoop,2], nCntFor
            ),1,,,"AC2",aMemoAC2[nLoop,1])
Next nLoop
Else
    If ( nCntFor <= Len(aRegistro) )
        dbDelete()
        MSMM(AC2->AC2_CODMEM,,,2)

        +-----+
        +
        | Exclui os campos memo de usuario |
        +-----+
        +
        For nLoop := 1 To Len( aMemoAC2 )
            MSMM(aMemoAC2[nLoop,1],,,,2)
        Next nLoop

    EndIf
EndIf
MsUnLock()
Next nCntFor
+-----+
| Exclusao |
+-----+
Otherwise
    For nCntFor := 1 To Len(aRegistro)
        AC2->(dbGoto(aRegistro[nCntFor]))
        RecLock("AC2")
        dbDelete()
        MsUnLock()
        MSMM(AC2->AC2_CODMEM,,,2)
    Next nCntFor

    If !Empty( Select( "AC9" ) )
        +-----+
        | Exclui a amarracao de conhecimento |
        +-----+
        MsDocument( "AC1", AC1->( Recno() ), 2, , 3 )
    EndIf

EndCase
+-----+
| Atualizacao do cabecalho |
+-----+
dbSelectArea("AC1")
dbSetOrder(1)
If ( MsSeek(xFilial("AC1")+M->AC1_PROVEN) )
    RecLock("AC1")
Else
    If ( lGravou )
        RecLock("AC1",.T.)
    EndIf
EndIf
If ( !lGravou )
    dbDelete()
    MSMM(AC1->AC1_CODMEM,,,2)
+-----+

```

```

| Exclui os campos memo de usuario |
+-----+
For nLoop := 1 To Len( aMemoAC1 )
    MSMM( AC1->( FieldGet( aMemoAC1[ nLoop, 1 ] ) ),,,2)
Next nLoop
Else
    For nCntFor := 1 To AC1->(FCount())
        If ( FieldName(nCntFor)!="AC1_FILIAL" )
            FieldPut( nCntFor,M->&(FieldName(nCntFor)))
        Else
            AC1->AC1_FILIAL := xFilial( "AC1" )
        EndIf
    Next nCntFor
    MSMM( AC1->AC1_CODMEM,,M->AC1_MEMO,1,,,"AC1", "AC1_CODMEM" )
    +-----+
    | Grava os campos memo de usuario |
    +-----+
    For nLoop := 1 To Len( aMemoAC1 )
        MSMM( AC1->( FieldGet( aMemoAC1[nLoop,1] ) ),,,;
        M->&( aMemoAC1[nLoop,2] ),1,,,"AC1",aMemoAC1[nLoop,1])
    Next nLoop
EndIf
MsUnlock()
+-----+
|   Restaura integridade da rotina   |
+-----+
RestArea(aArea)
Return( .T. )
/*
+-----+-----+-----+-----+-----+
| Funcao      |Ft010TudOK| Autor |Eduardo Riera      | Data |13.01.2000|
+-----+-----+-----+-----+-----+
| Descricao   |Funcao TudoOK|      |                    |      |
+-----+-----+-----+-----+-----+
| Sintaxe     | Ft010TudOK()|      |                    |      |
+-----+-----+-----+-----+-----+
| Parametros  | Nenhum      |      |                    |      |
+-----+-----+-----+-----+-----+
| Retorno     | .T../.F.    |      |                    |      |
+-----+-----+-----+-----+-----+
| Uso         | FATA010     |      |                    |      |
+-----+-----+-----+-----+-----+
*/
Function Ft010TudOk()
Local lRet      := .T.
Local nPosRelev := GDFieldPos( "AC2_RELEVA" )
Local nPosStage := GDFieldPos( "AC2_STAGE" )
Local nLoop     := 0
Local nTotal    := 0
Local nPosDel   := Len( aHeader ) + 1
If !Empty( AScan( aCols, { |x| x[nPosRelev] > 0 } ) )
    For nLoop := 1 To Len( aCols )
        If !aCols[ nLoop, nPosDel ]
            nTotal += aCols[ nLoop, nPosRelev ]
        Else
            +-----+
            | Permite excluir apenas se não estiver em uso por oportunidade |
            +-----+

            AD1->( dbSetOrder( 5 ) )
            If AD1->( dbSeek( xFilial( "AD1" ) + M->AC1_PROVEN +
            aCols[nLoop,nPosStage] ) )
                Aviso( STR0007, STR0011 + AllTrim( aCols[nLoop,nPosStage] ) + ;
                STR0012, { STR0009 }, 2 ) ;
                // Atencao // "A etapa " // " nao pode ser excluida pois esta em
                uso por uma ou mais // oportunidades !"
                lRet := .F.
            Exit

        EndIf
    Next nLoop
EndIf

```

```

        EndIf
    Next nLoop

    If lRet
        If nTotal <> 100
            Aviso( STR0007, STR0008, ;
                { STR0009 }, 2 ) //"Atencao !"###"A soma dos valores de relevancia
                deve ser igual a 100% //"###"Fechar"
            lRet := .F.
        EndIf
    EndIf
EndIf
Return( lRet )

```

```

/*
+-----+-----+-----+-----+-----+-----+
| Funcao   | Ft010DelOk | Autor | Sergio Silveira   | Data | 18.01.2001 |
+-----+-----+-----+-----+-----+-----+
| Descricao | Validacao da Exclusao |
+-----+-----+-----+-----+-----+-----+
| Sintaxe   | Ft010DelOk() |
+-----+-----+-----+-----+-----+-----+
| Parametros | Nenhum      |
+-----+-----+-----+-----+-----+-----+
| Retorno   | .T./.F.     |
+-----+-----+-----+-----+-----+-----+
| Uso       | FATA010     |
+-----+-----+-----+-----+-----+-----+
*/
Static Function Ft010DelOk()
LOCAL lRet := .T.
AD1->( dbSetOrder( 5 ) )
If AD1->( dbSeek( xFilial( "AD1" ) + M->AC1_PROVEN ) )
    lRet := .F.
    Aviso( STR0007, STR0010, { STR0009 }, 2 ) // "Atencao"
    // "Este processo de venda nao pode ser excluido pois esta sendo utilizado em uma
    // ou mais
    // oportunidades !", "Fechar"
EndIf
Return( lRet )

```

Modelos de Relatórios

Existem várias formas de se gerar um relatório no sistema, no entanto a forma de se elaborar o programa não varia muito. Abaixo mostramos um modelo-padrão, que utiliza as funções básicas na geração de um relatório.

```

/*
+-----+-----+-----+-----+-----+-----+
| Funcao   | MATR425   | Autor | Rodrigo de Sartorio | Data | 11/05/95   |
+-----+-----+-----+-----+-----+-----+
| Descricao | Relatorio de Estoque por Lote |
+-----+-----+-----+-----+-----+-----+
| Sintaxe   | MATR425() |
+-----+-----+-----+-----+-----+-----+
| Uso       | Generico  |
+-----+-----+-----+-----+-----+-----+
|           |           |           |           |           |
|           | ATUALIZACOES SOFRIDAS DESDE A CONSTRUCAO NICIAL |
|           |           |           |           |           |
+-----+-----+-----+-----+-----+-----+
| Programador | Data   | BOPS | Motivo da Alteracao |
+-----+-----+-----+-----+-----+-----+
|           |           |           |           |           |
+-----+-----+-----+-----+-----+-----+
*/
#include 'MATR425.CH'

```

```

#include 'FIVEWIN.CH'
Function MATR425()
+-----+
| Define Variaveis |
+-----+
Local cDesc1      := STR0001 //"Este programa emitira' uma relacao com a posiç,õ de "
Local cDesc2      := STR0002 //"estoque por Lote/Sub-Lote."
Local cDesc3      := ''
Local cString     := 'SB8'
Local Titulo      := STR0003 //"Posicao de Estoque por Lote/Sub-Lote"
Local Tamanho     := 'M'
Local wnRel       := 'MATR425'
+-----+
| Variaveis Tipo Private padrao de todos os relatorios |
+-----+
Private aOrd      := {STR0004,STR0005} //" Por Produto"###" Por Lote/Sub-Lote"
Private aReturn   := {STR0006,1,STR0007, 1, 2, 1, '',1 } //"Zebrado"###"Administracao"
Private cPerg     := 'MR425A'
Private nLastKey  := 0
Private nTipo     := 0
+-----+
| Verifica as perguntas selecionadas |
+-----+
Pergunte('MR425A', .F.)
+-----+
| Variaveis utilizadas para parametros |
| mv_par01      // Do Produto |
| mv_par02      // Ate Produto |
| mv_par03      // De Lote |
| mv_par04      // Ate Lote |
| mv_par05      // De Sub-Lote |
| mv_par06      // Ate Sub-Lote |
| mv_par07      // De Local |
| mv_par08      // Ate Local |
| mv_par09      // Lista Saldo Zerado ? Lista/Nao Lista |
| mv_par10      // Do Tipo |
| mv_par11      // Ate o Tipo |
| mv_par12      // Do Grupo |
| mv_par13      // Ate o Grupo |
+-----+
| Envia controle para SETPRINT |
+-----+
wnRel := SetPrint(cString,wnRel,cPerg,@Titulo,cDesc1,cDesc2,cDesc3,.F.,aOrd,,Tamanho)
nTipo := If(aReturn[4]=1,GetMv('MV_COMP'),GetMv('MV_NORM'))
If nLastKey == 27
    dbClearFilter()
    Return Nil
Endif
SetDefault(aReturn,cString)
If nLastKey == 27
    dbClearFilter()
    Return Nil
Endif
RptStatus({|lEnd| C425Imp(@lEnd,wnRel,Tamanho,Titulo)},Titulo)
Return Nil
/**
+-----+
| Funcao |C425Imp | Autor |Rodrigo Sartorio | Data | 14/11/95 |
+-----+
| Descricao | Chamada do Relatorio |
+-----+
| Uso | MATR425 |
+-----+
*/
Static Function C425Imp(lEnd, wnRel, Tamanho, Titulo)
+-----+
| Variaveis especificas dos relatorios |
+-----+
Local cIndex := ''
Local cCond := ''

```

```

Local cLoteAnt      := ''
Local cProdAnt      := ''
Local cDescAnt      := ''
Local cSLoteAnt     := ''
Local cAlmoAnt      := ''
Local cSeekSB8      := ''
Local cCondSB8      := ''
Local cNomArq       := ''
Local cPicSld       := PesqPict('SB8', 'B8_SALDO', 12)
Local cPicEmp       := PesqPict('SB8', 'B8_EMPENHO', 12)
Local dDataAnt      := CtoD(' / / ')
Local dValiAnt      := CtoD(' / / ')
Local nSaldo        := 0
Local nEmpenho      := 0
Local nSaldoT       := 0
Local nEmpenhoT     := 0
Local nCntImpr      := 0
Local nIndSB8       := 0
Local lSubLote      := .F.

+-----+
| Variaveis utilizadas para Impressao do Cabecalho e Rodape |
+-----+

Private aLinha      := {}
Private Cabec1      := ''
Private Cabec2      := ''
Private cBTxt       := Space(10)
Private cBCont      := 0
Private Li          := 80
Private M_PAG       := 01
/-- Condicao de Filtragem da IndRegua
cCond := 'B8_FILIAL=>' + xFilial('SB8') + '.And.'
cCond += 'B8_PRODUTO=>' + mv_par01 + '.And.B8_PRODUTO<=' + mv_par02 + '.And.'
cCond += 'B8_LOTECTL>=' + mv_par03 + '.And.B8_LOTECTL<=' + mv_par04 + '.And.'
cCond += 'B8_NUMLOTE>=' + mv_par05 + '.And.B8_NUMLOTE<=' + mv_par06 + '.And.'
cCond += 'B8_LOCAL>=' + mv_par07 + '.And.B8_LOCAL<=' + mv_par08 + ''
If aReturn[8] == 1
    cIndex := 'B8_FILIAL+B8_PRODUTO+B8_LOCAL+B8_LOTECTL+B8_NUMLOTE'
    Titulo := STR0008 //"POSICAO DE ESTOQUE POR LOTE/SUBLOTE (POR PRODUTO)"
    Cabec1 := STR0009 //"PRODUTO DESCRICAO SUB-LOTE LOTE AL SALDO EMPENHO DATA
        DATA "
    Cabec2 := STR0014 //"
                                                                VALIDADE "
ElseIf aReturn[8] == 2
    cIndex := 'B8_FILIAL+B8_LOTECTL+B8_NUMLOTE+B8_PRODUTO+B8_LOCAL'
    Titulo := STR0010 //"POSICAO DE ESTOQUE POR LOTE/SUB-LOTE (POR LOTE)"
    Cabec1 := STR0011 //"SUB-LOTE LOTE PRODUTO DESCRICAO AL SALDO EMPENHO DATA
        DATA "
    Cabec2 := STR0014 //"
                                                                VALIDADE "
EndIf

+-----+
| Pega o nome do arquivo de indice de trabalho |
+-----+
cNomArq := CriaTrab('', .F.)
/-- Seta a Ordem Correta no Arquivo SB1
dbSelectArea('SB1')
dbSetOrder(1)

+-----+
| Cria Indice de Trabalho |
+-----+
dbSelectArea('SB8')
IndRegua('SB8', cNomArq, cIndex,, cCond, STR0017) //"Selecionando Registros..."
#IFDEF TOP
dbSetIndex(cNomArq+OrdBagExt())
#ENDIF
dbGoTop()
SetRegua>LastRec())

+-----+
| Processa o Laco de impressao |
+-----+

```



```

Do While !Eof()
+-----+
| Cancela a impressao                                     |
+-----+
If lEnd
    @ Prow()+1, 001 PSay STR0012 //"CANCELADO PELO OPERADOR"
    Exit
EndIf
lSubLote := Rastro(B8_PRODUTO,'S')
/-- Define a Quebra por Produto ou Lote
If aReturn[8] == 1
    cSeekSB8 := B8_FILIAL+B8_PRODUTO+B8_LOCAL
    cCondSB8 := 'B8_FILIAL+B8_PRODUTO+B8_LOCAL'
Else
    cSeekSB8 :=
    B8_FILIAL+B8_LOTECTL+If(lSubLote,B8_NUMLOTE,'')+B8_PRODUTO+B8_LOCAL
    cCondSB8 :=
    'B8_FILIAL+B8_LOTECTL'+If(lSubLote,'B8_NUMLOTE+',')+B8_PRODUTO+B8_LOCAL'
EndIf
nSaldo      := 0
nEmpenho    := 0
nSaldoT     := 0
nEmpenhoT   := 0

/-- Processa o Laco da Quebra
Do While !Eof() .And. cSeekSB8 == &(cCondSB8)

    /-- Atualiza a Regua de Impressao
    IncRegua()

+-----+
| Cancela a Impressao                                     |
+-----+
If lEnd
    @ Prow()+1, 001 PSay STR0012 //"CANCELADO PELO OPERADOR"
    Exit
EndIf

/-- Saldo do Lote ou Lote/Sublote
nSaldo      += B8_SALDO
nEmpenho    += B8_EMPENHO

/-- Saldo Total da Quebra
nSaldoT     += B8_SALDO
nEmpenhoT   += B8_EMPENHO

/-- Posiciona-se na Descricao Correta do SB1
If !(cProdAnt=B8_PRODUTO)
    SB1->(dbSeek(xFilial('SB1')+SB8->B8_PRODUTO, .F.))
EndIf

If SB1->B1_TIPO < mv_par10 .Or. SB1->B1_TIPO > mv_par11
    dbSkip()
    Loop
EndIf

If SB1->B1_GRUPO < mv_par12 .Or. SB1->B1_GRUPO > mv_par13
    dbSkip()
    Loop
EndIf

/-- Salva Dados do Registro Atual / Passa para o Proximo Registro
cProdAnt := B8_PRODUTO
cDescAnt := SubS(SB1->B1_DESC,1,30)
cSlotAnt := If(lSubLote,B8_NUMLOTE,Space(Len(B8_NUMLOTE)))
cLoteAnt := B8_LOTECTL
cAlmoAnt := B8_LOCAL
dDataAnt := B8_DATA
dValiAnt := B8_DTVALID
dbSkip()

```

```

/-- Imprime Saldo do Lote ou Lote/Sublote
If !(cSeekSB8==&(cCondSB8)) .Or. lSubLote .Or. !(cLoteAnt==B8_LOTECTL)
  /-- Verifica se Lista Saldo Zerado
  If mv_par09==2 .And. QtdComp(nSaldo)==QtdComp(0)
    Loop
  EndIf
  If Li > 58
    Cabec(Titulo,Cabec1,Cabec2,wnRel,Tamanho,nTipo)
  EndIf
  nCntImpr ++
  If aReturn[8] == 1
    @ Li, 000 PSay cProdAnt
    @ Li, 016 PSay cDescAnt
    @ Li, 047 PSay cSLotAnt
    @ Li, 054 PSay cLoteAnt
  ElseIf aReturn[8] == 2
    @ Li, 000 PSay cSLotAnt
    @ Li, 007 PSay cLoteAnt
    @ Li, 018 PSay cProdAnt
    @ Li, 034 PSay cDescAnt
  EndIf
  @ Li, 065 PSay cAlmoAnt
  @ Li, 068 PSay nSaldo Picture cPicSld
  @ Li, 081 PSay nEmpenho Picture cPicEmp
  @ Li, 094 PSay dDataAnt
  @ Li, 105 PSay dValiAnt
  Li ++
  nSaldo := 0
  nEmpenho := 0
EndIf
EndDo

/-- Imprime Saldo Total da Quebra
If nCntImpr > 0
  If Li > 58
    Cabec(Titulo,Cabec1,Cabec2,wnRel,Tamanho,nTipo)
  EndIf
  @ Li, 000 PSay If(aReturn[8]==1,STR0013,If(lSubLote,STR0016,STR0015))
  /--Total do Produto - @ Li, 068 PSay nSaldoT Picture cPicSld
  @ Li, 081 PSay nEmpenhoT Picture cPicEmp
  Li++
  @ Li, 000 PSay __PrtThinLine()
  Li++
  nCntImpr := 0
  nSaldoT := 0
  nEmpenhoT := 0
EndIf
EndDo
If !(Li==80)
  Roda(cBCont,cBtxt,Tamanho)
EndIf
/-- Restaura a Integridade do SB8
dbSelectArea('SB8')
RetIndex('SB8')
dbClearFilter()
If File(cNomArq+OrdBagExt())
  fErase(cNomArq+OrdBagExt())
Endif
If aReturn[5] == 1
  Set Printer To
  dbCommitAll()
  OurSpool(wnRel)
Endif
MS_Flush()
Return Nil

```

Existem diversas formas de se gerar uma rotina de cálculo no sistema, no entanto o modelo básico sempre se mantém. Abaixo mostramos um modelo-padrão que utiliza as funções básicas para tal.

```

/* /
+-----+-----+-----+-----+-----+-----+
+ Funcao   | FINA210 | Autor | Wagner Xavier           | Data | 01/12/92 |
+-----+-----+-----+-----+-----+-----+
| Descricao | Recalcula saldos bancarios dentro de um determinado periodo |
+-----+-----+-----+-----+-----+-----+
| Sintaxe   | FINA210()
+-----+-----+-----+-----+-----+-----+
| Uso       | Generico
+-----+-----+-----+-----+-----+-----+
|          | ATUALIZACOES SOFRIDAS DESDE A CONSTRUCAO NICIAL
+-----+-----+-----+-----+-----+-----+
|Programador| Data    | BOPS | Motivo da Alteracao
+-----+-----+-----+-----+-----+-----+
|          |         |      |
+-----+-----+-----+-----+-----+-----+
#include "FINA210.CH"
#include "PROTHEUS.CH"
Function FinA210()
LOCAL nOpca :=0
+-----+-----+-----+-----+-----+-----+
| Define Variaveis
+-----+-----+-----+-----+-----+-----+
LOCAL oDlg, aSays:={}, aButtons:={}
Private cCadastro := OemToAnsi(STR0004) //"Reconciliacao de Saldos Bancarios"
Pergunte("FIN210",.F.)
AADD (aSays,;
OemToAnsi(STR0005))//"Este programa tem como objetivo recalculer e analisar os saldos"
AADD (aSays,;
OemToAnsi(STR0006)) //"Bancarios dia a dia de um determinado periodo ate a data base do
"
AADD (aSays,;
OemToAnsi(STR0007)) //"sistema. Utilizando no caso de haver necessidade de retroagir a
"
AADD (aSays,;
OemToAnsi(STR0008)) //"movimentacao bancaria. Use como referencia a data em que o saldo
"
AADD (aSays,;
OemToAnsi(STR0009)) //"ficou defasado. "
AADD(aButtons, { 1,.T.,{|o| nOpca:= 1,o:oWnd:End()}} )
AADD(aButtons, { 2,.T.,{|o| o:oWnd:End() }} )
AADD(aButtons, { 5,.T.,{| Pergunte("FIN210",.T. ) }} )
FormBatch( cCadastro, aSays, aButtons )
If nOpca == 1
    #IFDEF TOP
    If TcSrvType() == 'AS/400'
        Processa({|lEnd| FA210Proc()}) // Chamada da funcao de reconciliacao
    Else
        Processa({|lEnd| FA211Proc()}) // Chamada da funcao de reconciliacao
    Endif
    #ELSE
    Processa({|lEnd| FA210Proc()}) // Chamada da funcao de reconciliacao
    #ENDIF
Endif
Return
/* /
+-----+-----+-----+-----+-----+-----+
| Funcao   | Fa210Proc | Autor | Wagner Xavier           | Data | 01.12.1992|
+-----+-----+-----+-----+-----+-----+
| Descricao | Funcao de recalculo dos saldos bancarios - CODEBASE
+-----+-----+-----+-----+-----+-----+
| Sintaxe   | Fa210Proc ( )
+-----+-----+-----+-----+-----+-----+
| Parametros | Nenhum
+-----+-----+-----+-----+-----+-----+
| Retorno   | Nenhum
+-----+-----+-----+-----+-----+-----+

```

```

+-----+-----+
|  Uso          |  FINA210                                     |
+-----+-----+
/*
Function FA210Processa()
LOCAL nSaldoIni
LOCAL nEntradas
LOCAL nSaidas, nData, cCond
LOCAL dDataMovto
LOCAL cFil      := ""
LOCAL lAllFil   := .F.
LOCAL cChave
LOCAL cIndex    := ""
LOCAL lSaida    := .F.
+-----+-----+
| Variaveis utilizadas para parametros      |
| mv_par01 // Do Banco                      |
| mv_par02 // Ate o Banco                   |
| mv_par03 // Da Agencia                     |
| mv_par04 // Ate a Agencia                 |
| mv_par05 // Da Conta                      |
| mv_par06 // Ate a Conta                   |
| mv_par07 // A partir da Data              |
+-----+-----+
dbSelectArea( "SA6" )
dbSeek( cFilial+mv_par01 , .T.)
ProcRegua(RecCount())
If Empty(xFilial( "SA6" )) .AND. !Empty(xFilial("SE5"))
+-----+-----+
| Filtra o arquivo pro tipo e vencimento    |
+-----+-----+
dbSelectArea("SE5")
cIndex := CriaTrab(nil,.f.)
cChave  := "E5_BANCO+E5_AGENCIA+E5_CONTA+DTOS(E5_DTDISPO)"
cCond   := 'dtos(E5_DTDISPO)>=' + dtos(mv_par07) + ''
IndRegua("SE5",cIndex,cChave,,cCond,OemToAnsi(STR0015)) //"Selecionando
Registros..."
nIndexSE5 := RetIndex("SE5")
#IFDEF TOP
dbSetIndex(cIndex+OrdBagExt())
#ENDIF
dbSetOrder(nIndexSE5+1)
lSaida := .T.
dbGoTop()
lAllFil:= .T.
Else
dbSelectArea("SE5")
cIndex := CriaTrab(nil,.f.)
cChave  := "E5_FILIAL+E5_BANCO+E5_AGENCIA+E5_CONTA+DTOS(E5_DTDISPO)"
cCond   := 'dtos(E5_DTDISPO)>=' + dtos(mv_par07) + '' .and. E5_FILIAL ==
'' + xFilial("SE5") + ''
IndRegua("SE5",cIndex,cChave,,cCond,OemToAnsi(STR0015)) //"Selecionando
Registros..."
nIndexSE5 := RetIndex("SE5")
#IFDEF TOP
dbSetIndex(cIndex+OrdBagExt())
#ENDIF
dbSetOrder(nIndexSE5+1)
dbGoTop()
Endif
+-----+-----+
| Inicia recalculo dos saldos atraves da movimentacao bancaria |
+-----+-----+
dbSelectArea( "SA6" )
dbSeek( cFilial+mv_par01 , .T.)
While !Eof() .and. A6_FILIAL == cFilial .and. A6_COD <= mv_par02
IncProc()
// Alteracao para nao recalculer o saldo dos caixas do Loja, pois
// estes devem ser recalculados atraves da opcao "Recalculo de Caixa" - Adriano
dbSelectArea("SX5")
If (dbSeek(xFilial("SX5")+"23"+SA6->A6_COD)) .or. (SA6->A6_Cod == "CL1")

```

```

        dbSelectArea("SA6")
        dbSkip()
        Loop
    Endif
    dbSelectArea("SA6")
    cBanco := A6_COD
    cAgencia := A6_AGENCIA
    cConta := A6_NUMCON
    nSaldoIni:= 0
    nEntradas:= 0
    nSaidas := 0
    If cAgencia < mv_par03 .or. cAgencia > mv_par04 .or. cConta < mv_par05 .or. cConta
    > mv_par06
        dbSkip( )
        Loop
    Endif

+-----+
| Localiza Saldo de Partida. |
| Observe que o programa retorna um registro no banco de |
| dados, portanto a data de referencia , a data em que |
| o saldo ficou errado, nao a data correta do saldo. |
+-----+

    dbSelectArea( "SE8" )
    dbSeek( cFilial+cBanco+cAgencia+cConta+Dtos(mv_par07),.T. )
    dbSkip( -1 )
    If E8_BANCO != cBanco .or. E8_AGENCIA != cAgencia .or. E8_CONTA != cConta .or.
    BOF() .or. EOF()
        nSaldoIni := 0
    Else
        nSaldoIni := E8_SALATUA
    End

+-----+
| Localiza movimentacao bancaria |
+-----+

    dbSelectArea( "SE5" )
    dbSetOrder(nIndexSE5+1)
    cFil := Iif(!lAllFil,"",xFilial("SE5"))
    dbSeek(cFil+cBanco+cAgencia+cConta+Dtos(mv_par07),.T.)
    While !Eof() .and. E5_BANCO+E5_AGENCIA+E5_CONTA == cBanco+cAgencia+cConta
        IF !lAllFil .and. E5_FILIAL != xFilial("SE5")
            Exit
        Endif
        dDataMovto := E5_DTDISPO
        While !Eof() .and. E5_BANCO+E5_AGENCIA+E5_CONTA+dtos(E5_DTDISPO)== ;
            cBanco+cAgencia+cConta+dtos(dDataMovto)
            IF !lAllFil .and. E5_FILIAL != xFilial("SE5")
                Exit
            Endif
            IF E5_TIPODOC $ "DC/JR/MT/CM/D2/J2/M2/C2/V2/CP/TL" //Valores de Baixas
                dbSkip()
                Loop
            Endif
            If E5_VENCTO > E5_DATA // Ignora pre' datados - gerou titulo
                dbSkip()
                Loop
            Endif
            If E5_SITUACA = "C" //Cancelado
                dbSkip()
                Loop
            Endif
            If SE5->E5_MOEDA $ "C1/C2/C3/C4/C5" .and. Empty(SE5->E5_NUMCHEQ)
                dbSkip()
                Loop
            Endif

+-----+
| Na transferencia somente considera nestes numerarios |
| No Final00 , tratado desta forma. |
| As transferencias TR de titulos p/ Desconto/Cau#eo (FINA060) |
| n#o sofrem mesmo tratamento dos TR bancarias do FINA100 |
| Aclaracao : Foi incluido o tipo $ para os movimentos em di-- |
| nheiro em QUALQUER moeda, pois o R$ nao e representativo |
+-----+

```

```

| fora do BRASIL. |
+-----+
If SE5->E5_TIPODOC $ "TR/TE" .and. Empty(SE5->E5_NUMERO)
    If !(E5_MOEDA $ " $ /R$/DO/TB/TC/CH"+IIf(cPaisLoc="BRA","","/$
    "))
        dbSkip()
    Loop
Endif
Endif
If E5_TIPODOC $ "TR/TE" .and. (Substr(E5_NUMCHEQ,1,1)="*" ;
.or. Substr(E5_DOCUMEN,1,1) == "*" )
    dbSkip()
    Loop
Endif
If SE5->E5_MOEDA == "CH" .and. IsCaixaLoja(SE5->E5_BANCO) //Sangria
    dbSkip()
    Loop
Endif
If SubStr(E5_NUMCHEQ,1,1)="*" //cheque para juntar (PA)
    dbSkip()
    Loop
Endif
If !Empty(SE5->E5_MOTBX)
    If !MovBcoBx(SE5->E5_MOTBX)
        dbSkip()
    Loop
Endif
Endif
+-----+
| Baixa automatica |
+-----+
IF E5_TIPODOC = "BA"
    dbSkip()
    Loop
Endif
dbSelectArea("SE5")
IF E5_RECPAG = "R"
    nEntradas += E5_VALOR
Else
    nSaidas += E5_VALOR
Endif
dbSkip()
End // Quebra da data
dbSelectArea("SE8")
dbSeek(cFilial + cBanco+cAgencia+cConta+Dtos(dDataMovto))
IF Eof()
    RecLock("SE8",.t.)
Else
    RecLock("SE8",.f.)
Endif
Replace E8_FILIAL With cFilial,;
E8_BANCO With cBanco,;
E8_AGENCIA With cAgencia,;
E8_CONTA With cConta,;
E8_DTSALATU With dDataMovto
Replace E8_SALATUA With nSaldoIni+nEntradas-nSaidas
MsUnlock()
dbSelectArea("SE5")
IF !(SE5->(Eof())) .and. E5_BANCO+E5_AGENCIA+E5_CONTA ==
cBanco+cAgencia+cConta
    IF !AllFil .and. E5_FILIAL != xFilial("SE5")
        Exit
    Endif
For nData := dDataMovto+1 to ((SE5->E5_DTDISPO) - 1)
    dbSelectArea("SE8")
    If dbSeek(cFilial + cBanco+cAgencia+cConta+dtos(nData))
        RecLock("SE8",.F.,.T.)
        dbDelete()
        MsUnlock()
        SX2->(MsUnlock())
    EndIf

```

```

        dbSelectArea("SE5")
    Next
Endif
IF SE5->(Eof()) .or. E5_BANCO+E5_AGENCIA+E5_CONTA != cBanco+cAgencia+cConta
    dbSelectArea("SE8")
    dbSeek(cFilial + cBanco+cAgencia+cConta+dtos(dDataMovto+1),.t.)
    While !Eof() .and. cFilial + cBanco+cAgencia+cConta == ;
        E8_FILIAL+E8_BANCO+E8_AGENCIA+E8_CONTA
        RecLock("SE8",.F.,.T.)
        dbDelete()
        MsUnlock()
        SX2->(MsUnlock())
        dbselectArea("SE8")
        dbSkip()
    End
Endif
    dbSelectArea("SE5")
End // Fecha Primeiro Loop do SE5
dbSelectArea("SA6")
RecLock("SA6")
Replace A6_SALATU With nSaldoIni+nEntradas-nSaidas
MsUnlock()
dbSkip()
End
dbSelectArea("SE5")
RetIndex("SE5")
Set Filter To
If !Empty(cIndex)
    ferase( cIndex+OrdBagExt())
EndIf
Return NIL
*/
+-----+-----+-----+-----+-----+-----+
| Funcao      | Fa211Proc | Autor | Wagner Xavier          | Data | 09.03.2000 |
+-----+-----+-----+-----+-----+-----+
| Descricao   | Funcao de recalculo dos saldos bancarios - TOP CONNECT |
+-----+-----+-----+-----+-----+-----+
| Sintaxe     | Fa211Proc () |
+-----+-----+-----+-----+-----+-----+
| Parametros  | Nenhum |
+-----+-----+-----+-----+-----+-----+
| Retorno     | Nenhum |
+-----+-----+-----+-----+-----+-----+
| Uso         | FINA210 |
+-----+-----+-----+-----+-----+-----+
*/
#ifdef TOP
Function FA211Proc()
LOCAL nSaldoIni, nEntradas
LOCAL nSaidas, nData
LOCAL cQuery
LOCAL dDataMovto
+-----+
| Variaveis utilizadas para parametros |
| mv_par01 // Do Banco |
| mv_par02 // Ate o Banco |
| mv_par03 // Da Agencia |
| mv_par04 // Ate a Agencia |
| mv_par05 // Da Conta |
| mv_par06 // Ate a Conta |
| mv_par07 // A partir da Data |
+-----+
+-----+
| Inicia recalculo de saldos atraves da movimentacao bancaria |
+-----+
cQuery := "SELECT SA6.R_E_C_N_O_ A6_RECNO,"
cQuery += " E5_BANCO, E5_AGENCIA, E5_CONTA, E5_DTDISPO, E5_TIPODOC, E5_MOEDA,"
cQuery += " E5_NUMCHEQ, E5_MOTBX, E5_NUMERO, E5_RECPAG, E5_VALOR, E5_DOCUMEN"
cQuery += " FROM " + RetSqlName("SA6") + " SA6, " + RetSqlName("SE5") + " SE5"
cQuery += " WHERE A6_FILIAL = " + xFilial("SA6") + "'"
cQuery += " AND A6_COD between " + mv_par01 + " AND " + mv_par02 + "'"

```

```

cQuery += " AND A6_AGENCIA between '" + mv_par03 + "' AND '" + mv_par04 + "'"
cQuery += " AND A6_NUMCON between '" + mv_par05 + "' AND '" + mv_par06 + "'"
cQuery += " AND SA6.D_E_L_E_T_ <> '*' "
cQuery += " AND E5_VENCTO <= E5_DATA "
cQuery += " AND A6_COD = E5_BANCO "
cQuery += " AND A6_AGENCIA = E5_AGENCIA "
cQuery += " AND A6_NUMCON = E5_CONTA "
cQuery += " AND E5_SITUACA <> 'C' "
cQuery += " AND E5_TIPODOC <> 'BA' "
cQuery += " AND SE5.D_E_L_E_T_ <> '*' "
If Empty(xFilial( "SA6")) .AND. !Empty(xFilial("SE5"))
    cQuery += " AND E5_DTDISPO >= '" + dtos(mv_par07) + "'"
    //O filtro de filial entre ' ' e 'ZZ' foi adicionado para o SQL utilizar indice
    cQuery += " AND E5_FILIAL between ' ' AND 'ZZ' "
Else
    cQuery += " AND E5_DTDISPO >= '" + dtos(mv_par07) + "' AND E5_FILIAL = '" +
    xFilial("SE5") + "'"
Endif
cQuery += " ORDER BY E5_BANCO, E5_AGENCIA, E5_CONTA, E5_DTDISPO "

cQuery := ChangeQuery(cQuery)
dbUseArea(.T., "TOPCONN", TcGenQry(, cQuery), 'TRB210', .T., .T.)
TcSetField("TRB210", "E5_DTDISPO", "D")
While !Eof()
    dbSelectArea("TRB210")
    cBanco := E5_BANCO
    cAgencia := E5_AGENCIA
    cConta := E5_CONTA
    nSaldoIni := 0
    nEntradas := 0
    nSaidas := 0
    +-----+
    | Localiza Saldo de Partida. |
    | Observe que o programa retorna um registro no banco de |
    | dados, portanto a data de referencia , a data em que |
    | o saldo ficou errado, nao a data correta do saldo. |
    +-----+
    dbSelectArea( "SE8" )
    dbSeek( cFilial+cBanco+cAgencia+cConta+Dtos(mv_par07), .T. )
    dbSkip( -1 )
    If E8_BANCO != cBanco .or. E8_AGENCIA != cAgencia .or. E8_CONTA != cConta .or.
    BOF() .or. EOF()
        nSaldoIni := 0
    Else
        nSaldoIni := E8_SALATUA
    Endif
    +-----+
    | Localiza movimentacao bancaria |
    +-----+
    dbSelectArea("TRB210")
    While !Eof() .and. E5_BANCO+E5_AGENCIA+E5_CONTA == cBanco+cAgencia+cConta
        dDataMovto := E5_DTDISPO
        While !Eof() .and. E5_BANCO+E5_AGENCIA+E5_CONTA+DTOS(E5_DTDISPO) == ;
            cBanco+cAgencia+cConta+dtos(dDataMovto)
            IF E5_TIPODOC $ "DC/JR/MT/CM/D2/J2/M2/C2/V2/CP/TL" //Valores de Baixas
                dbSkip()
                Loop
            Endif
            If E5_MOEDA $ "C1/C2/C3/C4/C5" .and. Empty(E5_NUMCHEQ)
                dbSkip()
                Loop
            Endif
            +-----+
            | Na transferencia somente considera nestes numerarios |
            | No Final00 , tratado desta forma. |
            | As transferencias TR de titulos p/ Desconto/Cau#eo (FINA060) |
            | n#o sofrem mesmo tratamento dos TR bancarias do FINA100 |
            +-----+
            If E5_TIPODOC $ "TR/TE" .and. Empty(E5_NUMERO)
                If !(E5_MOEDA $ "R$/DO/TB/TC/CH")
                    dbSkip()

```



```

                Loop
            Endif
        Endif
    Endif
    If E5_TIPODOC $ "TR/TE" .and. (Substr(E5_NUMCHEQ,1,1)=="*" ;
    .or. Substr(E5_DOCUMEN,1,1) == "*" )
        dbSkip()
        Loop
    Endif
    If E5_MOEDA == "CH" .and. IsCaixaLoja(E5_BANCO) //Sangria
        dbSkip()
        Loop
    Endif
    If SubStr(E5_NUMCHEQ,1,1)=="*" //cheque para juntar (PA)
        dbSkip()
        Loop
    Endif
    If !Empty(E5_MOTBX)
        If !MovBcoBx(E5_MOTBX)
            dbSkip()
            Loop
        Endif
    Endif
    If E5_RECPAG = "R"
        nEntradas += E5_VALOR
    Else
        nSaidas += E5_VALOR
    Endif
    dbSkip()
End // Quebra da data
dbSelectArea("SE8")
dbSeek(cFilial+cBanco+cAgencia+cConta+Dtos(dDataMovto))
IF Eof()
    RecLock("SE8",.t.)
Else
    RecLock("SE8",.f.)
Endif
Replace E8_FILIAL With cFilial,;
E8_BANCO With cBanco,;
E8_AGENCIA With cAgencia,;
E8_CONTA With cConta,;
E8_DTSALATU With dDataMovto
Replace E8_SALATUA With nSaldoIni+nEntradas-nSaidas
MsUnlock()
dbSelectArea("TRB210")
IF !(TRB210->(Eof())) .and. E5_BANCO+E5_AGENCIA+E5_CONTA ==
cBanco+cAgencia+cConta
    For nData := dDataMovto+1 to ((E5_DTDISPO) - 1)
        dbSelectArea("SE8")
        If dbSeek(cFilial + cBanco+cAgencia+cConta+dtos(nData))
            RecLock("SE8",.F.,.T.)
            dbDelete()
            MsUnlock()
            SX2->(MsUnlock())
        EndIf
        dbSelectArea("TRB210")
    Next
Endif
If TRB210->(Eof()) .or. E5_BANCO+E5_AGENCIA+E5_CONTA !=
cBanco+cAgencia+cConta
    dbSelectArea("SE8")
    dbSeek(cFilial+cBanco+cAgencia+cConta+dtos(dDataMovto+1),.t.)
    While !Eof() .and. cFilial+cBanco+cAgencia+cConta == ;
        E8_FILIAL+E8_BANCO+E8_AGENCIA+E8_CONTA
        RecLock("SE8",.F.,.T.)
        dbDelete()
        MsUnlock()
        SX2->(MsUnlock())
        dbselectArea("SE8")
        dbSkip()
    Enddo
Endif

```

```
        dbSelectArea("TRB210")
    Enddo // Fecha Primeiro Loop do SE5
    dbSelectArea("SA6")
    If dbSeek( xFilial("SA6")+cBanco+cAgencia+cConta)
        RecLock("SA6")
        Replace A6_SALATU With nSaldoIni+nEntradas-nSaidas
        MsUnlock()
    EndIf
    dbSelectArea("TRB210")
    dbSkip()
Enddo
dbSelectArea("TRB210")
dbCloseArea()
Return NIL
#ENDIF
```

Funções básicas da Linguagem

AAdd

Adiciona um novo elemento ao final do array.

Sintaxe

`AADD(aAlvo, expValor) --> Valor`

Argumento	Obrigat.	Tipo	Descrição
<i>aAlvo</i>	Sim	A	É o array ao qual o novo elemento será adicionado.
<i>expValor</i>	Sim	Todos	É uma expressão válida que será o valor do novo elemento.

Retorno	Descrição
AADD()	avalia <i>expValue</i> e retorna seu Valor. Se <i>expValue</i> não for especificado, AADD() retorna NIL.

Descrição

AADD() é uma função de array que incrementa o tamanho real do array *aAlvo* em um. O elemento de array recentemente criado é atribuído o *Valor* especificado por *expValue*.

AADD() é usado para aumentar dinamicamente uma array. É útil para listas ou filas criadas dinamicamente.

Um exemplo bom deste é o GetList, array usada pelo sistema para armazenar objetos do tipo Get. Depois que um READ ou CLEAR GETS é executado, a GetList ficará vazia. Cada vez que você executa um comando @...GET, o sistema usa AADD() para adicionar um novo elemento à array GetList, e atribui então um novo objeto GET ao novo elemento.

AADD() é similar a ASIZE(), mas apenas adiciona um elemento por vez; ASIZE() pode incrementar ou decrementar uma array a um determinado tamanho. AADD(), entretanto, têm a vantagem de poder atribuir um valor ao elemento recentemente adicionado, enquanto que ASIZE() não pode. AADD() pode também parecer similar a AINS(), mas é diferente: AINS() move elementos dentro de uma disposição, mas os ele não muda o tamanho.

Se *expValue* for uma outra array, o novo elemento em *aAlvo* conterá uma referência à array especificada por *expValue*.

Exemplos

```
aArray := {} // Resultado: aArray is an empty array
AADD(aArray, 5) // Resultado: aArray is { 5 }
AADD(aArray, 10) // Resultado: aArray is { 5, 10 }
AADD(aArray, { 12, 10 }) // Resultado: aArray is
// { 5, 10, { 12, 10 } }
```

ACopy

Copia os elementos de uma array para outra.

Sintaxe

ACOPY(*aOrigem*, *aDestino*, [*nInicio*], [*nQtde*], [*nPosDestino*]) --> *aDestino*

Argumento	Obrigat.	Tipo	Descrição
<i>aOrigem</i>	Sim	A	é o array que contém os elementos a serem copiados.
<i>aDestino</i>	Sim	A	é o array que receberá a cópia dos elementos.
<i>nInicio</i>	Não	N	indica qual o índice do primeiro elemento de <i>aOrigem</i> que será copiado. Se não for especificado, o valor assumido será 01.
<i>nQtde</i>	Não	N	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> .
<i>nPosDestino</i>		N	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	Descrição
ACOPY()	retorna uma referência ao array <i>aDestino</i> .

Descrição

ACOPY() é uma 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. Para fazer isso, uso a função aClone().

Exemplos

```
Este exemplo cria dois arrays, com um conteúdo cada. Os dois primeiros elementos do array
fonte são então copiados no array destino:
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 }
```

AClone

Duplica arrays aninhadas ou multi-dimensionais.

Sintaxe

ACLONE(*aOrigem*) --> aDuplicado

Argumento	Obrigat.	Tipo	Descrição
<i>aOrigem</i>	Sim	A	É o array original que será duplicado.

Retorno	Descrição
aDuplicado	Array idêntico ao aOrigem, porem sem nenhuma referência.

Descrição

ACLONE() é uma função de array, que cria duplicatas completas do array *aSource*. Se *aSource* contiver arrays aninhadas, ACLONE() criará o mesmo padrão de aninhamento preenchendo com cópias dos valores contidos em *aSource*.

ACLONE() é similar a ACOPY(), exceto que ACOPY() não duplica arrays aninhadas.

Ao igualarmos dois arrays, eles ficam associados por referência, utilizando aClone() não existe referência.

Exemplos

```
LOCAL aOne, aTwo, aThree
aOne := { 1, 2, 3 } // Resultado: aOne is {1, 2, 3}
aTwo := ACLONE(aOne) // Resultado: aTwo is {1, 2, 3}
aThree := aOne // Resultado: aThree is {1, 2, 3}
aOne[1] := 99 // Result: aOne is {99, 2, 3}
// aTwo continua {1, 2, 3} mais o aThree fica { 99, 2, 3 }
```

ADel

Exclui um elemento de um array.

Sintaxe

ADEL(*aOrigem*, *nPos*) --> *aOrigem*

Argumento	Obrigat.	Tipo	Descrição
aOrigem	Sim	A	É o array de onde será excluído um item
nPos	Sim	A	É a posição a partir da 1, do qual será excluído um elemento

Retorno	Descrição
ADEL()	Retorna uma referência ao <i>aOrigem</i> .

Descrição

ADEL() é uma função de manipulação que elimina uma posição do array, deslocando as posições posteriores. A última posição do array passa a ter o conteúdo NIL.

Caso a posição a ser eliminada seja um array, este será eliminado.

Exemplos

```
LOCAL aArray
aArray := { 1, 2, 3 } // Resultado: { 1, 2, 3 }
ADEL(aArray, 2) // Resultado: { 1, 3, NIL }
```

ADir

Preenche vários arrays com informações de arquivos e diretórios.

Sintaxe

ADIR([*cArqEspec*], [*aNomeArq*], [*aTamanho*], [*aData*], [*aHora*], [*aAtributo*]) --> Numérico

Argumento	Obrigat.	Tipo	Descrição
<i>cArqEspec</i>	Não	C	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 (*.*).
<i>aNomeArq</i>	Não	A	Array de Caracteres. É o array com os nomes dos arquivos encontrados na busca. O conteúdo anterior do array é apagado.
<i>aTamanho</i>	Não	A	Array Numérico. São os tamanhos dos arquivos encontrados na busca.
<i>aData</i>	Não	A	Array de Datas. São as datas de modificação dos arquivos encontrados na busca.
<i>aHora</i>	Não	A	Array de Caracteres. São os horários de modificação dos arquivos encontrados. Cada elemento contém horário no formato: hh:mm:ss.
<i>aAtributos</i>	Não	A	Array de Caracteres. São os atributos dos arquivos, caso esse array seja passado como parâmetros, serão incluídos os arquivos com atributos de Sistema, Ocultos.

Retorno	Descrição
> = 0	Quantidade de arquivos encontrados

Descrição

ADir() preenche os arrays passados com os dados dos arquivos encontrados, através da máscara informada. Tanto arquivos locais (Remote) como do servidor podem ser informados. ADir é uma função obsoleta, utilize sempre Directory().

Exemplos

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

AEval

Executa um code block para cada elemento de um array.

Sintaxe

AEVAL(*aArray*, *bBloco*, [*nInicio*], [*nQtde*]) --> *aArray*

Argumento	Obrigat.	Tipo	Descrição
<i>aArray</i>	Sim	A	É o array que será atravessado pelo bloco
<i>bBlock</i>	Sim	Bloco de código	É o bloco que será executado para cada elemento do Array.
<i>nInicio</i>	Não	N	É a posição inicial. Se não for especificada o início será a partir do 1°.
<i>nQtde</i>	Não	N	É o numero de elementos que devem ser processados a partir de <i>nInicio</i> . O Valor padrão são todos os elementos do <i>nInicio</i> até o final.

Retorno	Descrição
AEVAL()	Retorna uma referência de <i>aArray</i> .

Descrição

AEVAL() executa um code block para cada elemento de um array, passando cada um como o parâmetro do bloco. É muito semelhante ao DBEVAL().

AEVAL() passa cada elemento de um array para o code block sem se preocupar com o tipo.

Exemplos

```
// Exemplo 1
#include "Directry.ch"
//
LOCAL aFiles := DIRECTORY("*.dbf"), nTotal := 0
AEVAL(aFiles,{ | aDbfFile | QOUT(PADR(aDbfFile[F_NAME], 10), aDbfFile[F_SIZE]),;
nTotal += aDbfFile[F_SIZE]);
} )
//
? "Total Bytes:", nTotal
// Exemplo 2
#include "Directry.ch"
//
LOCAL aFiles := DIRECTORY("*.dbf"), aNames := {}
AEVAL(aFiles, { | file | AADD(aNames, file[F_NAME]) } )
//Exemplo 3
LOCAL aArray[6]
AFILL(aArray,"old")
AEVAL(aArray,;
{|cValue,nIndex| IF(cValue == "old",;
aArray[nIndex] := "new",)})
```


EVAL()

DBEVAL()

AFill

Preenche um array com um determinado valor.

Sintaxe

AFill(aDestino , expValor, [nInicio], [nQuantidade]) --> aDestino

Argumento	Obrigat.	Tipo	Descrição
aDestino	Sim	A	É o onde os dados serão preenchidos.
expValor	Sim	Todos	É o dado que será preenchido em todas as posições informadas, não é permitida a utilização de arrays.
[nInicio]	Não	N	É a posição inicial de onde os dados serão preenchidos, o valor padrão é 1.
[nCount]	Não	N	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	Descrição
AFill()	Retorna uma referência para aDestino.

Descrição

AFill() é função de manipulação de arrays, que preenche os elementos do array com qualquer tipo de dado. Incluindo code block.

Afill() não deve ser usado para preencher um array com outro array.

Exemplos

```
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. }
```

Alns

Insere um elemento com conteúdo NIL em um array.

Sintaxe

`AINS(aOrigem, nPos) --> alns`

Argumento	Obrigat.	Tipo	Descrição
aOrigem	Sim	A	É o array de onde será inserido um item.
nPos	Sim	A	É a posição a partir da 1, do qual será inserido um elemento

Retorno	Descrição
AINS()	Retorna uma referência ao aOrigem.

Descrição

AINS() é um função de manipulação de array que insere um elemento na posição determinada por *nPos*.

O novo elemento possui conteúdo igual a NIL. Após a inserção, o último elemento será excluído.

Para alterar o tamanho de um array, utilize `aSize()`.

Exemplos

```
LOCAL aArray
aArray := { 1, 2, 3 } // Resultado: { 1, 2, 3 }
AINS(aArray, 2) // Resultado: { 1, NIL, 2 }
```

Alias

Verifica qual é o Alias.

Sintaxe

`ALIAS([nAreaTrab]) --> Caracter`

Argumento	Obrigat.	Tipo	Descrição
nAreaTrab	Não	N	Número da área de trabalho a ser verificada.

Retorno	Descrição
""	Não existe tabela em uso na área de trabalho verificada.
cAlias	String contendo o Alias da área de trabalho verificada (em maiúsculo).

Descrição

Verifica qual é o Alias de determinada área de trabalho; se ela não foi especificada, será verificado qual 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.

Exemplo

```
// Este exemplo mostra como o Alias corrente pode ser apresentado para o usuário.
dbUseArea( .T., "dbfcdxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F. )
MessageBox( "O Alias corrente é: " + Alias(), "Alias", 0) // Resultado: "O Alias
corrente é: SSS"
```

Append From

Importa registros de outra tabela ou arquivo texto.

Sintaxe

APPEND FROM Arquivo [FIELDS Campos] [FOR CondFor] [WHILE CondWhile] [NEXT nRecs] [RECORD nRecno] [REST] [ALL] [VIA RDD] [SDF | DELIMITED [WITH BLANK | cDelimitador]]

Argumento	Obrigat.	Tipo	Descrição
Arquivo	Sim		Nome do arquivo cujos registros serão importados, pode ser apenas o nome ou na forma de uma string.
Campos	Não		Lista dos campos a serem copiados, se omitido serão copiados todos os campos.
CondFor	Não		Expressão em ADVPL a ser resolvida para que o registro seja copiado.
CondWhile	Não		Expressão em ADVPL que determina quando a cópia deve parar (quando a expressão retornar

			.F.).
nRecs	Não	N	Quantos registros devem ser copiados.
nRecno	Não	N	Número do recno do registro a ser copiado.
RDD	Não		Nome do RDD utilizado na importação (entre aspas simples ou dupla), se omitido será utilizado o RDD corrente.
cDelimitador	Não	C	Especifica qual caracter foi utilizado como delimitador para gerar o arquivo do tipo texto.

Descrição

Este comando é utilizado para copiar alguns registros do arquivo especificado por "FROM *cArquivo*" utilizando-se o driver especificado por "VIA *RDD*" se especificado.

Se forem especificados os campos de interesse através de "FIELDS *Campos*", apenas eles serão copiados, caso contrário todos campos o serão. Se não forem especificadas as condições para que o registro seja copiado, copia toda a tabela como se estivesse especificada a opção "ALL".

Pode-se especificar um escopo para que os registros sejam copiados através das opções "FOR *CondFor*" e "WHILE *CondWhile*". Pode-se também limitar o número de registros a serem copiados através da opção "NEXT *nRecs*" e determinar que a cópia dos registros deve-se iniciar a partir do registro atual com "REST", mas caso contrário o comando executa um [DBGOTOP\(\)](#) antes de iniciar a cópia. Se é desejado copiar apenas determinado registro pode-se defini-lo através da especificação do recno com "RECORD *nRecno*".

Além de copiar registros de uma tabela normal, pode-se copiar registros de um arquivo texto que contenha os dados desejados. Existem duas formas de gerar estes arquivos textos com [COPY TO](#), utilizando-se o SDF que trabalha com tamanhos de registros e campos fixos (preenchendo com espaços) ou através de um delimitador que separa os campos",". Deve-se especificar na cópia qual o tipo do arquivo texto através das opções "SDF" ou "DELIMITED".

Pode-se ainda especificar qual o delimitador utilizado nas strings da seguinte forma:

- "DELIMITED WITH BLANK" - as strings não possuem delimitador;
- "DELIMITED WITH *cDelimitador*" - as strings são delimitadas com o caracter especificado;

Caso seja omitido o delimitador será considerado o delimitador padrão (" ").

Exemplo

```
// Este exemplo demonstra como utilizar o comando APPEND FROM para acrescentar alguns
campos de registros (apenas 10 registros) de outra tabela pertencentes a um escopo
definido a partir do registro atual para a tabela corrente:
USE Cliente VIA "CTRECDX" NEW
```

```

APPEND FROM Amigos FIELDS Nome,Nascimento,End,Tel FOR Idade>20 WHILE Nome<"VVV" NEXT 10
REST
// Este exemplo demonstra como se pode utilizar o comando <@>COPY TO para gravar os dados
de uma tabela em um arquivo do tipo texto (DELIMITED) e incluí-los a outra tabela com o
comando APPEND FROM:
USE Amigos NEW
COPY TO temp DELIMITED // Registro: "Jose",19751108,69411233,12.00
COPY TO temp1 DELIMITED WITH BLANK // Registro: Jose 19751108 69411233 12.00
COPY TO temp2 DELIMITED WITH "@" // Registro: @Jose@,19751108,69411233,12.00
USE Cliente NEW
APPEND FROM temp DELIMITED
USE Clientel NEW
APPEND FROM temp1 DELIMITED WITH BLANK
USE Cliente2 NEW
APPEND FROM temp2 DELIMITED WITH "@"
// Este exemplo demonstra como se pode utilizar o comando <@>COPY TO para gravar os dados
de uma tabela em um arquivo do tipo texto (SDF) e incluí-los a outra tabela com o comando
APPEND FROM:
USE Amigos NEW
COPY TO temp3 SDF // Registro: Jose      1975110869411233  12.00
USE Cliente3 NEW
APPEND FROM temp3 SDF

```

Array

Cria um array com dados não inicializados.

Sintaxe

ARRAY(*nQtdElementos1* , [*nQtdElementosn*]...) --> aArray

Argumento	Obrigat.	Tipo	Descrição
<i>nQtdElementos1</i>	Sim	N	Quantidade de Elementos da 1ª dimensão do array.
[<i>nQtdElementosN</i>]	Não	N	Quantidade de Elementos das demais dimensões do array.

Retorno	Descrição
ARRAY()	Retorna um array com as dimensões especificadas.

Descrição

Array() é uma função que retorna um array não inicializado com múltiplas dimensões.

Se mais de um argumento for especificado, será retornado um array multidimensional

A vantagem de utilizar ARRAY(), ao invés de outras opções, é a possibilidade de usá-la em code blocks e expressões.

Exemplo

```
// Exemplo 1 - Dois métodos idênticos.
aArray := ARRAY(5)
aArray := { NIL, NIL, NIL, NIL, NIL }
// Métodos equivalentes.
aArray := ARRAY(3, 2)
aArray := { {NIL, NIL}, {NIL, NIL}, {NIL, NIL} }
aArray := { ARRAY(2), ARRAY(2), ARRAY(2) }
```

AScan

Busca em um array até que o bloco retorne verdadeiro .T.

Sintaxe

ASCAN(aOrigem, expSearch, [nStart], [nCount]) --> nStoppedAt

Argumento	Obrigat.	Tipo	Descrição
aOrigem	Sim	A	É o array onde será executada a expressão.
<expSearch>	Sim	Todos	É a posição a partir da 1, do qual será inserido um elemento

Argumento Descrição

<aOrigem> Obrigatório, Array. É o array onde será executada a expressão.

<expSearch> Obrigatório, Qualquer Tipo. Casis either a simple value to scan for, or a code block.

If <expSearch> is a simple value it can be character, date, logical, or numeric type.

<nStart> is the starting element of the scan. If this argument is not specified, the default starting position is one.

<nCount> is the number of elements to scan from the starting position. If this argument is not specified, all elements from the starting element to the end of the array are scanned.

Returns

ASCAN() returns a numeric value representing the array position of the last element scanned. If <expSearch> is a simple value, ASCAN() returns the position of the first matching element, or zero if a match is not found. If <expSearch> is a code block, ASCAN() returns the position of the element where the block returned true (.T.).

Description

ASCAN() is an array function that scans an array for a specified value and operates like SEEK when searching for a simple value. The <expSearch> value is compared to the target array element beginning with the leftmost character in the target element and proceeding until there are no more characters left in <expSearch>. If there is no match, ASCAN() proceeds to the next element in the array. Since ASCAN() uses the equal operator (=) for comparisons, it is insensitive to the status of EXACT. If EXACT is ON, the target array element must be exactly equal to the result of <expSearch> to match. If the <expSearch> argument is a code block, ASCAN() scans the <aTarget> array executing the block for each element accessed. As each element is encountered, ASCAN() passes the element's value as an argument to the code block, and then performs an EVAL() on the block. The scanning operation stops when the code block returns true (.T.), or ASCAN() reaches the last element in the array.

Examples

This example demonstrates scanning a three-element array using simple values and a code block as search criteria. The code block criteria shows how to perform a case-insensitive search:

```
aArray := { "Tom", "Mary", "Sue" }
? ASCAN(aArray, "Mary") // Result: 2
? ASCAN(aArray, "mary") // Result: 0
//
? ASCAN(aArray, { |x| UPPER(x) ;
== "MARY" }) // Result: 2
```

This example demonstrates scanning for multiple instances of a search argument after a match is found:

```
LOCAL aArray := { "Tom", "Mary", "Sue", ;
"Mary" }, nStart := 1
//
// Get last array element position
nAtEnd := LEN(aArray)
DO WHILE (nPos := ASCAN(aArray, "Mary", ;
nStart)) > 0
? nPos, aArray[nPos]
//
// Get new starting position and test
// boundary condition
IF (nStart := ++nPos) > nAtEnd
EXIT
ENDIF
ENDDO
```

This example scans a two-dimensional array using a code block.

Note that the parameter aVal in the code block is an array:

```
LOCAL aArr:={}  
  
CLS  
  
AADD(aArr,{"one","two"})  
AADD(aArr,{"three","four"})  
AADD(aArr,{"five","six"})  
  
? ASCAN(aArr, { |aVal| aVal[2] == "four"}) // Returns 2
```

See Also

AEVAL()

EVAL()

ASize

Acrescenta ou reduz elementos de um array.

Sintaxe

ASIZE(*aOrigem* , *nNovoTamanho*) --> *aOrigem*

Argumento	Obrigat.	Tipo	Descrição
aOrigem	Sim	A	É o array que será modificado.
nNovoTamanho	Sim	N	É o novo tamanho do Array. Caso seja maior do que tamanho atual, os novos elementos ficarão com conteúdo NIL.

Retorno	Descrição
ASIZE()	Retorna uma referência ao array <i>aOrigem</i> .

Descrição

ASIZE() é uma função de manipulação de array que aumenta ou diminui o tamanho de um array.

No caso de diminuição, os últimos elementos serão apagados, em caso de aumento de tamanho, os novos elementos terão conteúdo NIL.

Exemplos

```
aArray := { 1 } // Resultado: aArray is { 1 }
```



```
ASIZE(aArray, 3) // Resultado: aArray is { 1, NIL, NIL }
ASIZE(aArray, 1) // Resultado: aArray is { 1 }
```

ASort

Ordena um array.

Sintaxe

ASORT(*aOrigem*, [*nInicio*], [*nQtde*], [*bOrdem*]) --> *aOrigem*

Argumento	Obrigat.	Tipo	Descrição
<i>aOrigem</i>	Sim	A	É o array que será classificado.
<i>nInicio</i>	Não	N	Onde será o início da ordenação. Caso seja omitido, será considerado o 1º elemento do array.
<i>nQtde</i>	Não	N	Quantidade de elementos que serão ordenados a partir do <i>nInicio</i> . Caso seja omitido, serão considerados todos elementos até o final do Array.
<i>bOrder</i>	Não	Bloco de código	É um bloco de código (code block) opcional que indicará a ordem correta dos elementos. Caso ele não seja informado, o array será classificado em ordem crescente desde que não seja multidimensional.

Retorno	Descrição
ASORT()	Retorna uma referência ao array <i>aOrigem</i> .

Descrição

ASORT() é uma função de manipulação de array que classifica todo ou parcialmente um array de tipo simples. Os tipos que podem ser classificados automaticamente são: caracter, data, lógico e numérico.

Caso *bOrder* seja especificado, o code block será usado para ordenar o array, neste caso, pode-se usar dados de qualquer tipo. Cada vez que o bloco for avaliado, serão passados dois elementos do array, caso eles estejam em ordem, deve-se retornar .T.

Exemplos

```
// Exemplo 1
aArray := { 3, 5, 1, 2, 4 }
ASORT(aArray)
```

```
// Resultado: { 1, 2, 3, 4, 5 }
ASORT(aArray,,, { |x, y| x > y })
// Resultado: { 5, 4, 3, 2, 1 }
//Exemplo 2
aArray := { "Fred", Kate", "ALVIN", "friend" }
ASORT(aArray,,, { |x, y| UPPER(x) < UPPER(y) })
//Exemplo 3
aKids := { {"Mary", 14}, {"Joe", 23}, {"Art", 16} }
aSortKids := ASORT(aKids,,, { |x, y| x[2] < y[2] })
Resultado:
{ {"Mary", 14}, {"Art", 16}, {"Joe", 23} }
```

ATail

Retorna o último elemento de um array

Sintaxe

ATAIL(*aArray*) --> Element

Argumento	Obrigat.	Tipo	Descrição
aArray	Sim	A	É o array de onde será retornado o último elemento.

Retorno	Descrição
ATAIL()	Retorna o último elemento de um array.

Descrição.

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*)]

Exemplos

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

BTVCanOpen

Verifica se a tabela BTrieve pode ser aberta.

Sintaxe

BTVCanOpen(*cNome* , [*cIndice*])->Lógico

Argumento	Obrigat.	Tipo	Descrição
cNome	Sim	C	Nome da tabela a ser testada.
cIndice	Não	C	Nome do arquivo de índice da tabela a ser testada.

Retorno	Descrição
.F.	Não é possível abrir a tabela testada. Principais motivos: Não existe o arquivo da tabela ou do índice fisicamente, ou as definições da tabela ou índice em questão não foram encontradas.
.T.	A tabela testada pode ser aberta.

Descrição

Esta função verifica se a tabela definida pelo parâmetro *cNome* pode ser aberta e, se existir, o parâmetro *cIndice* verifica, também, se o índice pode ser aberto. Para tanto, é testado se os arquivos envolvidos existem fisicamente, caso afirmativo, é verificado se as definições envolvidas são encontradas nos arquivos do DDF's.

Exemplo

```
// Este exemplo demonstra o uso típico de BTVCanOpen(). Se não falhar, a tabela e o
// índice testados serão abertos. Se falhar, uma mensagem é apresentada.
IF ! BTVCanOpen( "\dadosadv\aa1990.dat", "\dadosadv\ind1.ind" )
    MessageBox( "Não é possível abrir a tabela testada", "Erro", 0 )
    BREAK
ELSE
    Use "\dadosadv\aa1990.dat" SHARED NEW
    OrdListAdd( "\dadosadv\ind1.ind" )
ENDIF
```

BTVCreateDDFs

Gera os arquivos de definições (DDF's) compatíveis com outras ferramentas que manipulam arquivos Btrieve (Crystal Reports).

Sintaxe

BTVCreateDDFs (*aTabelas* , [*cDiretorio*]) -> Lógico

Argumento	Obrigat.	Tipo	Descrição
<i>aTabelas</i>	Sim	A	Nomes das tabelas e os respectivos diretórios (opcional).
<i>cDiretorio</i>	Não	C	Nome do diretório (abaixo do root) onde serão criados os novos DDF's.

Retorno	Descrição
.F.	Não conseguiu gerar os novos arquivos de definição. Principais erros: RDD não é Btrieve; diretório não está dentro do Protheus; não pode carregar as informações de definição ou não pode gravar os novos arquivos de definição.
.T.	Transformação de definições ocorrida com sucesso.

Descrição

Esta função transforma as informações armazenadas nos arquivos DDF's para o padrão utilizado por outras ferramentas, principalmente para geração de relatórios. Sendo que podem ser selecionadas apenas as tabelas de interesse através do parâmetro *aTabelas*.

Ex: `aTabelas := { {"AA3990", "C:\DADOS"}, {"AA4990", "C:\DADOS1"}, {"AA5990"} }`

•

Se o diretório não for especificado, será utilizado o diretório definido no arquivo FILE.BTV.

Os novos arquivos de definição, FILE.DDF, FIELD.DDF e INDEX.DDF, são gerados no diretório especificado pelo parâmetro *cDiretório*, se ele for omitido, serão gerados no mesmo diretório dos SXs.

Exemplo

```
// Este exemplo demonstra o uso típico de BTVCreateDDFs(). Se não falhar, serão gerados
os novos arquivos de definição. Se falhar, uma mensagem é apresentada.
b:= { {"AA3990"}, {"SA1990", "c:\protheus507\dadosadv"} }
IF !BTVCreateDDFs(b, "\temp") // Será concatenado com o RootPath
    MessageBox("Não foi possível montar o array com os nomes das tabelas","Erro", 0)
    BREAK
ENDIF
```

BTVDropIdxs

Apaga os índices da tabela corrente.

Sintaxe

`BTVDropIdxs () -> Lógico`

Retorno	Descrição
.F.	Não conseguiu apagar os índices. Principais erros: RDD não é Btrieve, não achou as definições no DDF, o arquivo não está exclusivo

.T. Deleção de índices ocorrida com sucesso

Descrição

A função BTVDropIdxs apaga os índices da tabela corrente, com exceção do índice interno, apenas se o mesmo for Btrieve e estiver aberto exclusivo. Para tanto ela executa os seguintes passos:

1. Fecha todos os índices;
2. Apaga as definições dos índices nos arquivos do diretório DDF;
3. Apaga os índices do arquivo da tabela corrente. Todos os índices criados de forma permanente ficam guardados na estrutura da tabela. Quando a tabela for aberta, todos os índices criados de forma permanente e o índice interno serão abertos também. Por isso, é recomendada a criação de índices de forma temporária.

Exemplo

```
// Este exemplo demonstra o uso típico de BTVDropIdxs(). Se não falhar, os índices são
// apagados e o processo continua. Se falhar, uma mensagem é apresentada.
USE Clientes SHARED NEW
IF !BTVDropIdxs()
    MessageBox("Não foi possível deletar os índices da tabela corrente","Erro", 0)
    BREAK
ENDIF
```

BTVTables

Retorna array composto por nomes das tabelas definidas no DDF do Protheus (FILE.BTV).

Sintaxe

BTVTables ()-->Array

Retorno	Descrição
NIL	Não conseguiu montar o array. Principais erros: RDD não é Btrieve ou não conseguiu recuperar as informações corretamente do arquivo FILE.BTV do DDFs.
Array	Lista com os nomes das tabelas extraídas do DDF.

Descrição

Verifica todos os nomes das tabelas armazenados no arquivo FILE.BTV do DDF e retorna um array com todos eles. Toda tabela criada possui o nome acrescentado neste arquivo de definições.

Exemplo

```
// Este exemplo demonstra o uso típico de BTVTables(). Se não falhar, é montado um array
// com os nomes das tabelas e esses nomes são mostrados no servidor. Se falhar, uma mensagem
// é apresentada.
```

```
a:= BTVTabl es()  
IF a=Nil  
    MessageBox("Não foi possível montar o array com os nomes das tabelas","Erro", 0)  
    BREAK  
ELSE  
    FOR i:= 1 to LEN(a)  
        ConOut(a[i])  
    NEXT  
ENDIF
```

CDow

Converte uma data para uma cadeia de caracteres contendo o dia da semana.

Sintaxe

CDOW(dExp) --> Caracter

Argumento	Obrigat.	Tipo	Descrição
dExp	Sim	D	É a data a converter.

Retorno	Descrição
Caracter	Nome do dia da semana como uma cadeia de caracteres. A primeira letra é maiúscula e as demais minúsculas.
""	Caso a data seja inválida ou nula

Descrição

CDOW() é uma função que converte uma data para uma cadeia de caracteres.

Exemplos

```
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

Converte uma data para uma cadeia de caracteres contendo o nome do mês.

Sintaxe

CMONTH(dData) --> Caracter

Argumento	Obrigat.	Tipo	Descrição
dData	S	D	É a data a converter.

Retorno	Descrição
Caracter	Retorna o nome do mês em uma cadeia de caracteres. A primeira letra do retorno em maiúscula e o restante do nome, em minúsculas.
""	Caso a data seja inválida ou nula.

Descrição

CMONTH() é uma função de conversão de datas que retorna uma cadeia de caracteres com o nome do mês em inglês.

Exemplos

Estes exemplos ilustram CMONTH():

```
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
```

Commit

Salva em disco as modificações de todas as tabelas.

Sintaxe

COMMIT

Descrição

Este comando salva em disco todas as atualizações pendentes em todas as áreas de trabalho.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o COMMIT para salvar todas as alterações realizadas nas áreas de trabalho abertas no momento.
USE Clientes NEW
DBGOTO(100)
Nome := "Jose"
USE Fornecedores NEW
DBGOTO(168)
Nome := "Joao"
COMMIT // Salva em disco as alterações realizadas nas tabelas Clientes e Fornecedores
```

Copy To

Copia registros da tabela corrente para uma nova tabela.

Sintaxe

COPY TO *Arquivo* [FIELDS *Campos*] [FOR *CondFor*] [WHILE *CondWhile*] [NEXT *nRecs*] [RECORD *nRecno*] [REST] [ALL] [VIA *RDD*] [SDF | DELIMITED [WITH BLANK | *cDelimitador*]

Argumento	Obrigat.	Tipo	Descrição
Arquivo	Sim		Nome do arquivo cujos registros serão exportados, pode ser apenas o nome ou na forma de uma string
Campos	Não		Lista dos campos a serem copiados, se omitido serão copiados todos os campos.
CondFor	Não		Expressão em ADVPL a ser resolvida para que o registro seja copiado.
CondWhile	Não		Expressão em ADVPL que determina quando a cópia deve parar (quando a expressão retornar .F.).
nRecs	Não	N	Quando registros devem ser copiados.
nRecno	Não	N	Número do recno do registro a ser copiado.
RDD	Não		Nome do RDD utilizado na importação (entre aspas simples ou dupla), se omitido será utilizado o RDD corrente.
cDelimitador	Não	C	Especifica qual caracter foi utilizado como delimitador para gerar o arquivo do tipo texto.

Descrição

Este comando é utilizado para copiar alguns registros da tabela corrente para o arquivo especificado por "TO *cArquivo*" utilizando-se o driver especificado por "VIA *RDD*" se especificado. Se forem especificados os campos de interesse através de "FIELDS *Campos*" apenas eles serão copiados, caso contrário todos campos o serão. Se não forem especificadas as condições para que o registro seja copiado, copia toda a tabela como se estivesse especificada a opção "ALL".

Pode-se especificar um escopo para que os registros sejam copiados através das opções "FOR *CondFor*" e "WHILE *CondWhile*". Pode-se também limitar o número de registros a serem copiados através da opção "NEXT *nRecs*" e determinar que a cópia dos registros deve-se iniciar a partir do registro atual com "REST", mas caso contrário o comando executa um [DBGOTOP\(\)](#) antes de iniciar a cópia. Se é desejado copiar apenas determinado registro pode-se defini-lo através da especificação do recno com "RECORD *nRecno*".

Além de copiar registros para uma tabela normal, pode-se copiar registros para um arquivo texto que contenha os dados desejados. Existem duas formas de gerar estes arquivos textos com [COPY TO](#), utilizando-se o SDF que trabalha com tamanhos de registros e campos fixos (preenchendo com espaços) ou através de um delimitador que separa os campos". Deve-se especificar na cópia qual o tipo do arquivo texto através das opções "SDF" ou "DELIMITED". Pode-se ainda especificar qual o delimitador utilizado nas strings da seguinte forma:

- "DELIMITED WITH BLANK" - as strings não possuem delimitador;
- "DELIMITED WITH *cDelimitador*" - as strings são delimitadas com o caracter especificado;

Caso seja omitido, o delimitador será considerado o delimitador padrão (" ").

Exemplo

```
// Este exemplo demonstra como utilizar o comando COPY TO criar nova tabela com alguns
campos escolhidos e alguns registros (apenas 10) da tabela atual pertencentes a um escopo
definido a partir do registro atual para determinada tabela:
USE Cliente VIA "CTRECDX" NEW
COPY TO Amigos FIELDS Nome,Nascimento,End,Tel FOR Idade>20 WHILE Nome<"VVV" NEXT 10 REST
// Este exemplo demonstra como se pode utilizar o comando COPY TO para gravar os dados de
uma tabela em um arquivo do tipo texto (DELIMITED) e incluí-los a outra tabela com o
comando APPEND FROM:
USE Amigos NEW
COPY TO temp DELIMITED // Registro: "Jose",19751108,69411233,12.00
COPY TO temp1 DELIMITED WITH BLANK // Registro: Jose 19751108 69411233 12.00
COPY TO temp2 DELIMITED WITH "@" // Registro: @Jose@,19751108,69411233,12.00
USE Cliente NEW
APPEND FROM temp DELIMITED
USE Clientel NEW
APPEND FROM temp1 DELIMITED WITH BLANK
USE Cliente2 NEW
APPEND FROM temp2 DELIMITED WITH "@"
// Este exemplo demonstra como se pode utilizar o comando <@>COPY TO para gravar os dados
de uma tabela em um arquivo do tipo texto (SDF) e incluí-los a outra tabela com o comando
APPEND FROM:
USE Amigos NEW
COPY TO temp3 SDF // Registro: Jose      1975110869411233  12.00
USE Cliente3 NEW
APPEND FROM temp3 SDF
```

Copy File

Copia Arquivos.

Sintaxe

CopyFile(*cOrigem*, *cDestino*) --> NIL

Argumento	Obrigat.	Tipo	Descrição
cOrigem	Sim	C	Nomes dos arquivos a serem copiados, aceita tanto arquivos locais (Cliente), como arquivos que estão

			no Servidor. WildCards são aceitos normalmente.
cDestino	Sim	C	Diretório com o destino dos arquivos a serem copiados, podendo também ser no servidor ou no cliente (estação local).

Retorno	Descrição
NIL	Sem retorno.

Descrição

Copia um arquivo, da origem para o destino, os caracteres * e ?, são aceitos normalmente.

Caso a origem esteja no cliente e o destino no servidor os arquivos são copiados para o servidor, o contrário também é valido.

Exemplo

```
__CopyFile( "C:\TEMP\*.DBF", "\BKP\*.DBF" ) // Copia arquivos do cliente para o Servidor
```

```
__CopyFile( "\TEMP\*.DBF", "\BKP\*.DBF" ) // Copia arquivos no servidor
```

Copy Structure

Copia a estrutura da tabela corrente para uma nova tabela.

Sintaxe

COPY STRUCTURE [FIELDS *Campos*] TO *Arquivo*

Argumento	Obrigat.	Tipo	Descrição
Campos	Não		Lista dos campos a serem incluídos na nova tabela, se omitido serão incluídos todos os campos.
Arquivo	Sim	C	Nome do arquivo da nova tabela a ser criada.

Descrição

Este comando é utilizado para criar nova tabela copiando a estrutura da tabela atual, sendo que pode-se selecionar apenas os campos de interesse através da opção "FIELD *Campos*" que se omitida, deixa que o comando copie toda a estrutura.

Semelhante ao funcionamento da função [DBCCREATE](#) com a passagem de parâmetro DBSTRUCT.

Exemplo

```
// Este exemplo demonstra como utilizar o comando COPY STRUCTURE no seu modo mais usual:
USE Cliente NEW
COPY STRUCTURE FIELDS Nome,Idade TO NovoCliente
// Este exemplo demonstra como o comando COPY STRUCTURE pode substituir o DBCREATE:
USE Cliente NEW
COPY STRUCTURE TO NovoCliente
// *** é semelhante a ***
USE Cliente NEW
DBCREATE("NovoCliente",DBSTRUCT())
```

CPYS2T

Copia arquivos do servidor para o cliente (Remote).

Sintaxe

CpyS2T(*cOrigem*, *cDestino*, [*ICompacta*]) --> NIL

Argumento	Obrigat.	Tipo	Descrição
cOrigem	Sim	C	Nomes dos arquivos a serem copiados, aceita apenas arquivos no servidor, WildCards são aceitos normalmente.
cDestino	Sim	C	Diretório com o destino dos arquivos no remote (Cliente).
ICompacta	Não	L	Indica se a cópia deve ser feita compactando o arquivo antes.

Retorno	Descrição
.T.	Arquivo foi copiado para o cliente com sucesso
.F.	Erro na cópia do Arquivo.

Descrição

Copia um arquivo, do servidor para o cliente (Remote), os caracteres * e ?, são aceitos normalmente.

Caso a compactação seja habilitada (*ICompacta*), os dados serão transmitidos de maneira compactada e descompactados antes do uso.

Exemplo

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

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

CPYT2S

Copia Arquivos entre o Cliente (Terminal) para o servidor.

Sintaxe

```
CpyT2S( cOrigem, cDestino, [ ICompacta ] ) --> NIL
```

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

Retorno	Descrição
.T.	Arquivo foi copiado para o cliente com sucesso.
.F.	Erro na cópia do Arquivo.

Descrição

Copia um arquivo, do cliente (Remote) para o servidor, os caracteres * e ?, são aceitos normalmente.

Caso a compactação seja habilitada (*ICompacta*), os dados serão transmitidos de maneira compacta e descompactados antes do uso.

Exemplo

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

CTreeDelldxs

Deleta os índices da tabela corrente.

Sintaxe

CtreeDelIdx() -> Lógico

Retorno	Descrição
.F.	<i>Não conseguiu deletar os índices. Principais erros: RDD não é Ctree; não fechou a tabela; não apagou o arquivo de índice; não atualizou as informações da tabela; não abriu a tabela novamente.</i>
.T.	Deleção de índices ocorrida com sucesso.

Descrição

A função CtreeDelIdx apaga os índices da tabela corrente, com exceção do índice interno, apenas se o mesmo for Ctree e estiver exclusiva. Para tanto, ela executa os seguintes passos:

1. Fecha os índices abertos;
2. Fecha a tabela;
3. Deleta os arquivos de índice fisicamente;
4. Atualiza as informações da tabela, removendo os índices de sua estrutura;
5. Abre novamente a tabela.

Todos os índices criados de forma permanente ficam guardados na estrutura da tabela. Portanto, não adianta deletar os arquivos de índices, pois quando a tabela for aberta, todos os índices criados de forma permanente e o índice interno serão recriados fisicamente (se não existirem); caso contrário, a tabela não será aberta. Por isso, é recomendada a criação de índices de forma temporária.

Exemplo

// Este exemplo demonstra o uso típico de CtreeDelIdx(). Se não falhar, os índices são apagados e o processo continua. Se falhar, uma mensagem é apresentada.

```
USE Clientes SHARED NEW
IF !CtreeDelIdx()
    MessageBox("Não foi possível deletar os índices da tabela corrente","Erro", 0)
    BREAK
ENDIF
```

CtreeDelInt

Deleta índice interno da tabela CTree.

Sintaxe

CtreeDelInt(cNome) -> Lógico

Argumento	Obrigat.	Tipo	Descrição
cNome	Sim	C	Especifica o nome da tabela cujo índice interno deve ser deletado.

Retorno	Descrição
.F.	Não conseguiu deletar o índice interno. Principais erros: tabela não está dentro do diretório do Protheus; não abriu a tabela ou não deletou o arquivo de índice interno.
.T.	Deleção do índice interno ocorrida com sucesso.

Descrição

A função CtreeDelInt apaga o índice interno de tabela Ctree, estando a mesma fechada. Para tanto, são executados os seguintes procedimentos:

1. Abre a tabela especificada pelo parâmetro cNome;
2. Verifica o nome do arquivo do índice interno na tabela;
3. Fecha a tabela;
4. Deleta fisicamente o arquivo do índice interno.

A tabela deve ser apagada após a chamada desta função, pois a tabela CTree não pode ser aberta sem índice interno.

Exemplo

```
// Este exemplo demonstra o uso típico de CtreeDelInt(). Sendo que a tabela
"\DADOSADV\SA1990.DTC" deve estar fechada. Se não falhar, o índice interno é apagado e o
processo continua. Se falhar, uma mensagem é apresentada.
IF !CtreeDelInt("\dadosadv\sa1990.dtc")
    MessageBox("Não foi possível deletar o índice da tabela","Erro", 0)
    BREAK
ENDIF
fErase("\dadosadv\sa1990.dtc")
```

CurDir

Retorna o diretório corrente.

Sintaxe

CURDIR([cNovoDir]) --> cDirAtual

Argumento	Obrigat.	Tipo	Descrição
cNovoDir	Não	C	Caminho com o novo diretório que será ajustado como corrente.

Retorno	Descrição
cDirAtual	Diretório corrente, sem a última barra.

Descrição

Retorna o diretório corrente do servidor, caso seja passado um caminho como parâmetro, esse diretório passará a ser o default.

Exemplo

```
? CURDIR("C:\TEMP")
```

Date

Retorna a data do sistema.

Sintaxe

DATE() --> Data

Retorno	Descrição
Data	<i>Data do sistema.</i>

Descrição

DATE() é a função que retorna a data do atual sistema. O formato de saída é controlado pelo comando SET DATE. O formato padrão é mm/dd/yy.

Exemplos

Estes exemplos mostram como usar a função DATE():

```
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

Retorna o dia do mês como valor numérico.

Sintaxe

DAY(dData) -->Numérico

Argumento	Obrigat.	Tipo	Descrição
dData	Sim	D	É a data a converter.

Retorno	Descrição
>=0 e <=31	Se o mês do argumento dData for fevereiro, anos bissextos são considerados.
0	Se a data do argumento dData for 29 de fevereiro e o ano não for bissexto, ou se o argumento dData for vazio.

Descrição

DAY() é uma 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.

Exemplos

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
```

DBAppend

Acrescenta um novo registro na tabela corrente.

Sintaxe

DBAppend ([lLiberaBloqueios]) ->Nil

Argumento	Obrigat.	Tipo	Descrição
lLiberaBloqueios	Não	L	Libera todos os registros bloqueados (locks), valor padrão é .T.

Descrição

Esta função acrescenta mais um registro em branco no final da tabela corrente, sempre é acrescentado e bloqueado. Se o parâmetro estiver com valor .T., todos os bloqueios de registros anteriores são liberados para que o novo registro seja acrescentado, caso contrário, se for .F., todos os bloqueios anteriores são mantidos.

Se este parâmetro não for especificado, o valor padrão é .T..

Exemplo

```
// Este exemplo demonstra como se pode utilizar o dbappend liberando e mantendo bloqueios anteriores.
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
```

DBCclearAllFilter

Limpa a condição de filtro de todas as ordens da lista.

Sintaxe

DBCclearAllFilter() -> Nil

Retorno	Descrição
NIL	Nenhum

Descrição

Esta função salva as atualizações realizadas e pendentes de todas as tabelas e depois limpa as condições de filtro de todas as ordens incluídas na lista. Seu funcionamento é oposto ao comando SET FILTER.

Exemplo

```
// Este exemplo demonstra como se pode utilizar DBCLEARALLFILTER para limpar a expressão de filtro.
USE Clientes NEW
DBSETFILTER( {| Idade < 40}, "Idade < 40" ) // Seta a expressão de filtro
...
DBCLEARALLFILTER() // Limpa a expressão de filtro de todas as ordens
```

DBCclearFilter

Limpa a condição de filtro.

Sintaxe

DBCclearFilter () ->Nil

Retorno	Descrição
NIL	Nenhum

Descrição

Esta função 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](#).

Exemplo

```
// Este exemplo demonstra como se pode utilizar DBCLEARFILTER para limpar a expressão de
filtro.
USE Clientes NEW
DBSETFILTER( {| Idade < 40}, "Idade < 40" ) // Seta a expressão de filtro
...
DBCLEARFILTER() // Limpa a expressão de filtro
```

DBCcloseAll

Fecha todas as tabelas abertas.

Sintaxe

DBCcloseAll () ->Nil

Retorno	Descrição
NIL	Nenhum

Descrição

Esta função 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.

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
```

DBCcloseArea

Fecha a área de trabalho.

Sintaxe

DBCcloseArea () ->Nil

Retorno	Descrição
NIL	Nenhum

Descrição

Esta função salva as atualizações pendentes na tabela corrente, libera todos os registros bloqueados e fecha a tabela corrente (área de trabalho). Seu funcionamento é semelhante ao comando CLOSE e é oposto à função DBUSEAREA e ao comando [USE](#).

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBCLOSEAREA para fechar a área de
trabalho atual.
USE Clientes NEW
DBSETINDEX("Nome") // Abre o arquivo de índice "Nome"
...
DBCLOSEAREA() // Fecha a área de trabalho atual, todos os índices e ordens
```

DBCclearIndex

Fecha todos os arquivos de índice da área de trabalho.

Sintaxe

DBCclearIndex () ->Nil

Retorno	Descrição
NIL	Nenhum

Descrição

Esta função 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.

Exemplo

```
// Este exemplo demonstra como se pode utilizar a função DBCLEARINDEX para fechar os índices.  
USE Clientes NEW  
DBSETINDEX("Nome") // Abre o arquivo de índice "Nome"  
...  
DBCLEARINDEX() // Fecha todos os arquivos de índices
```

DBCommit

Salva em disco todas as modificações da tabela corrente.

Sintaxe

DBCommit() ->Nil

Retorno	Descrição
NIL	Nenhum

Descrição

Esta função salva em disco todas as atualizações pendentes na área de trabalho corrente.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBCOMMIT para salvar todas as alterações realizadas na área de trabalho atual.  
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

Salva em disco todas as modificações.

Sintaxe

DBCommitAll () ->Nil

Retorno	Descrição
NIL	Nenhum

Descrição

Esta função salva em disco todas as atualizações pendentes em todas as áreas de trabalho.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBCOMMITALL para salvar todas as
alterações realizadas nas áreas de trabalho abertas no momento.
USE Clientes NEW
DBGOTO(100)
Nome := "Jose"
USE Fornecedores NEW
DBGOTO(168)
Nome := "Joao"
DBCOMMITALL() // Salva em disco as alterações realizadas nas tabelas Clientes e
Fornecedores
```

DBCreate

Cria nova tabela.

Sintaxe

DBCREATE(cNome , aEstrutura , [cDriver]) --> Nil

Argumento	Obrigat.	Tipo	Descrição
cNome	S	C	Nome do arquivo da tabela a ser criada (abaixo do "RootPath").
aEstrutura	S	Array	Lista com as informações dos campos para ser criada a tabela.
cDriver	N	C	Nome do RDD a ser utilizado para a criação da tabela. Se for omitido será criada com o corrente.

Retorno	Descrição
NIL	Nenhum

Descrição

Esta função 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}, como visto no exemplo a seguir.

Exemplo

```
// Este exemplo mostra como se pode criar novo arquivo de tabela 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}}
DBCREATE("\teste\amigos.xxx",aEstrutura) // Cria a tabela com o RDD corrente
USE "\teste\amigos.xxx" VIA "DBFCDX" NEW
```

DBCreatelndex

Cria um arquivo de índice.

Sintaxe

DBCREATEINDEX(cNome , cExpChave , [bExpChave], [lUnico]) --> Nil

Argumento	Obrigat.	Tipo	Descrição
cNome	S	C	Nome do arquivo de índice a ser criado.
cExpChave	S	C	Expressão das chaves do índice a ser criado na forma de string.
bExpChave	N	Bloco de Código	Expressão das chaves do índice a ser criado na forma executável.
lUnico	N	L	Cria índice como único (o padrão é .F.).

Retorno	Descrição
Nil	Não há tabela corrente ou a posição do campo especificado está inválida. Informação do campo requisitada pelo usuário (pode ser de tipo numérico se for tamanho ou casas decimais, tipo caracter se for nome ou tipo).

Descrição

Esta função 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:

1. Salva fisicamente as alterações ocorridas na tabela corrente;
2. Fecha todos os arquivos de índice abertos;
3. Cria o novo índice;

4. Seta o novo índice como a ordem corrente;
5. 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.

Exemplo

```
// Este exemplo mostra como se pode criar novo arquivo de índice criando a ordem sobre os campos Nome e End e não aceitará duplicação:  
USE Cliente VIA "DBFCDX" NEW  
DBCREATEINDEX ("teste\ind2.cdx", "Nome+End", { || Nome+End }, .T.)
```

DBDelete

Marca o arquivo corrente como deletado.

Sintaxe

DBDelete () ->Nil

Retorno	Descrição
NIL	Nenhum

Descrição

Esta função marca o arquivo corrente como deletado de modo semelhante ao comando [DELETE](#). Para filtrar os arquivos marcados pode-se utilizar o comando SET DELETED e para deletá-los fisicamente pode-se utilizar o comando [PACK](#).

Exemplo

```
// Este exemplo demonstra como se pode utilizar a função DBDELETE() para marcar alguns registros como deletados e o PACK para deletá-los fisicamente.  
USE Clientes NEW  
DBGOTO(100)  
DBDELETE()  
DBGOTO(105)  
DBDELETE()  
DBGOTO(110)  
DBDELETE()  
PACK
```

DBEval

Executa uma expressão para os registros dentro das condições especificadas.

Sintaxe

DBEVAL(*bBloco*, [*bForCond*], [*bWhileCond*], [*nProxRegs*], [*nRecno*], [*IRestante*]) --> Nil

Argumento	Obrigat.	Tipo	Descrição
bBloco	S	Bloco de Código	Expressão na forma executável a ser resolvida para cada registro processado.
bForCond	N	Bloco de Código	Expressão na forma executável a ser resolvida para verificar se o registro em questão está dentro do escopo definido.
bWhileCond	N	Bloco de Código	Expressão na forma executável a ser resolvida para verificar até qual registro será processado (até o bloco retornar .F.).
nProxRegs	N	N	Número de registros a ser processado a partir do registro corrente.
nRecno	N	N	Identificação de determinado registro a ser resolvida a expressão (recno).
IRestante	N	L	Processa o restante dos registros.

Descrição

Esta função é utilizada para executar uma expressão definida pelo bloco de código do primeiro parâmetro para cada registro que está dentro do escopo definido através dos blocos de condição de "for" e "while".

O número de registros a ser executado será definido com o parâmetro *nProxRegs* ou se setado o parâmetro *IRestante* serão executados todos os registros a partir do registro corrente até o final da tabela corrente. Se for especificado o parâmetro *nRecno* apenas o registro com o recno especificado será processado.

Se forem omitidos os blocos de "for" e "while", os mesmos serão considerados .T. como padrão, estão assim todos os registros dentro do escopo.

Se o parâmetro *IRestante* for omitido a tabela inicia o processamento dos registros a partir do topo da tabela, caso contrário serão processados os registros a partir do posicionamento corrente da tabela.

Exemplo

```
// Este exemplo mostra como se pode usar o DBEVAL para contar quantos registros estão
dentro do escopo especificado em toda a tabela, pois como o parâmetro IRestante foi
omitido a tabela irá para o topo antes de iniciar a processar os registros. Supondo que a
tabela está sobre um índice no campo idade, serão processados registros com o Nome cuja
ordem alfabética é maior que "FFFFF" e até encontrar algum registro de idade igual a 40:
USE Cliente VIA "DBFCDX" NEW
LOCAL nCount := 0;
DBGOTO(100)
DBEVAL( {|| nCount++}, {|| Nome > "FFFFF"}, {|| Idade < 40})
// Este exemplo mostra como se pode usar o DBEVAL para contar quantos registros estão
dentro do escopo especificado (como o exemplo anterior) a partir do registro atual (100):
USE Cliente VIA "DBFCDX" NEW
LOCAL nCount := 0;
```



```

DBGOTO(100)
DBEVAL( {|| nCount++}, {|| Nome > "FFFFF"}, {|| Idade < 40},,,.T.)
// Este exemplo mostra como se pode usar o DBEVAL para colocar numa variável um nome
inicial que está definido em um registro de recno definido (100):
USE Cliente VIA "DBFCDX" NEW
LOCAL cNomeIni := "";
DBEVAL( {|| cNomeIni := Nome},,,100)
// Este exemplo mostra como se pode usar o DBEVAL para verificar qual é o recno do décimo
registro a partir do corrente dentro do escopo definido:
USE Cliente VIA "DBFCDX" NEW
LOCAL nRecno := 0;
DBGOTO(100)
DBEVAL( {|| nRecno := RECNO()}, {|| Nome > "FFFFF"}, {|| Idade < 40},10,,.T.)

```

DBF

Verifica qual é o Alias corrente.

Sintaxe

DBF() --> Caracter

Retorno	Descrição
""	Não existe tabela em uso.
cAlias	String contendo o Alias corrente.

Descrição

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.

Exemplo

```

// Este exemplo mostra como o DBF corrente pode ser mostrado para o usuário.
dbUseArea( .T., "dbfcdxads", "\dadosadv609\sai990.dbf", "SSS", .T., .F. )
MessageBox("O Alias corrente é: " + DBF(), "Alias", 0) // Resultado: "O Alias
corrente é: SSS"

```

DBFieldInfo

Verifica informações sobre um campo.

Sintaxe

DBFieldInfo (*nInfoTipo* , *nCampo*) -> Informação do Campo

Argumento	Obrigat.	Tipo	Descrição
nInfoTipo	S	N	Tipo de informação a ser verificada.
nCampo	S	N	Posição do campo a ser verificado.

Retorno	Descrição
Nil	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).

Descrição

Esta função 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:

DBS_DEC	Número de casas decimais (tipo numérico)
DBS_LEN	Tamanho (tipo numérico)
DBS_TYPE	Tipo (tipo caracter)

A posição do campo não leva em consideração os campos internos do Protheus (recno e deleted).

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBFIELDINFO para obter as informações
do primeiro campo da tabela Clientes.
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

Verifica a expressão de filtro corrente.

Sintaxe

DBFilter () --> Caracter

Retorno	Descrição
""	Não existe filtro ativo na área de trabalho atual.
cExpFiltro	Expressão do filtro ativo na área de trabalho atual.

Descrição

Esta função é utilizada para verificar a expressão de filtro ativo na área de trabalho corrente.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBFILTER para verificar a expressão do
filtro corrente.
USE Cliente INDEX Ind1 NEW
SET FILTER TO Nome > "Jose"
DBFILTER() // retorna: Nome > "Jose"
SET FILTER TO Num < 1000
DBFILTER() // retorna: Num < 1000
```

DBGoBottom

Posiciona no último registro lógico.

Sintaxe

DBGoBottom() -> Nil

Retorno	Descrição
NIL	Nenhum

Descrição

Esta função é utilizada para posicionar a tabela corrente no último registro lógico. A sequência lógica depende da ordem e do filtro ativo sobre a tabela corrente, portanto o último registro lógico pode não ser o último registro físico.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBGOBOTTOM para posicionar no último
registro físico.
USE Cliente
DBGOBOTTOM() // Posiciona no último registro físico, pois não há ordem ativa
// Este exemplo demonstra como se pode utilizar o DBGOBOTTOM para posicionar no último
registro lógico.
USE Cliente INDEX Ind1 NEW
DBGOBOTTOM() // Posiciona no último registro lógico (último registro na sequência gerada
pelo índice)
```

DBGoTo

Posiciona em determinado registro.

Sintaxe

DBGoTo(*nRegistro*) -> Nil

Argumento	Obrigat.	Tipo	Descrição
nRegistro	S	N	Número do registro a ser posicionado.

Descrição

Esta função é utilizada para posicionar a tabela corrente em determinado registro, segundo a ordem física (seqüência sobre o recno).

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBGOTO para posicionar a tabela  
corrente em determinado registro.  
USE Cliente INDEX Ind1 NEW  
DBGOTO(100) // Posiciona no registro de recno 100
```

DBGoTop

Posiciona no primeiro registro lógico.

Sintaxe

DBGoTop() -> Nil

Retorno	Descrição
NIL	Nenhum

Descrição

Esta função é utilizada para posicionar a tabela corrente no primeiro registro lógico. A seqüência lógica depende da ordem e do filtro ativo sobre a tabela corrente, portanto o primeiro registro lógico pode não ser o primeiro registro físico.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBGOTOP para posicionar no primeiro
registro físico.
USE Cliente
DBGOTOP() // Posiciona no primeiro registro físico, pois não há ordem ativa
// Este exemplo demonstra como se pode utilizar o DBGOTOP para posicionar no primeiro
registro lógico.
USE Cliente INDEX Ind1 NEW
DBGOTOP() // Posiciona no primeiro registro lógico (primeiro registro na sequência gerada
pelo índice)
```

DBInfo

Verifica informações sobre a tabela corrente.

Sintaxe

DBInfo (*nInfoTipo*) -> Informação da Tabela

Argumento	Obrigat.	Tipo	Descrição
nInfoTipo	S	N	Tipo de informação a ser verificada.

Retorno	Descrição
<i>Nil</i>	<p>Não há tabela corrente.</p> <p>Informação da Tabela Informação requisitada pelo usuário (o tipo depende da informação requisitada).</p>

Descrição

Esta função é utilizada para obter informações sobre a tabela corrente. O tipo de informação (primeiro argumento) é escolhido de acordo com as constantes abaixo:

DBI_GETRECSIZE	Tamanho do registro em número de bytes similar a RECSIZE (tipo numérico)
DBI_TABLEEXT	Extensão do arquivo da tabela corrente (tipo caracter)
DBI_FULLPATH	Nome da tabela corrente com caminho completo (tipo caracter)
DBI_BOF	Verifica se está posicionada no início da tabela similar a BOF (tipo lógico)
DBI_EOF	Verifica se está posicionada no final da tabela similar a EOF (tipo lógico)
DBI_FOUND	Verifica se a tabela está posicionada após uma pesquisa similar a FOUND (tipo lógico)
DBI_FCOUNT	Número de campos na estrutura da tabela corrente similar a FCOUNT (tipo

	numérico)
DBI_ALIAS	Nome do Alias da área de trabalho corrente similar a ALIAS (tipo caracter)
DBI_LASTUPDATE	Verifica a data da última modificação similar a LUPDATE (tipo data)

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBINFO para obter as informações da
tabela corrente (Clientes).
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.
```

DBOrderInfo

Verifica informações sobre uma ordem.

Sintaxe

DBOrderInfo (*nInfoTipo* , [*cIndice*] , [*cOrdem* | *nPosicao*]) -> Informação da Ordem

Argumento	Obrigat.	Tipo	Descrição
nInfoTipo	S	N	Nome do arquivo de índice.
cOrdem	N	C	Nome da ordem.
nPosição	N	N	Posição da ordem na lista de ordens ativas.

Retorno	Descrição
Nil	Não há ordem corrente ou a posição da ordem especificada está inválida. Informação da Ordem Informação requisitada pelo usuário (pode ser de tipo numérico se for número de ordens no índice, tipo caracter se for nome do arquivo de índice).

Descrição

Esta função 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:

DBOI_BAGNAME	Nome do arquivo de índice ao qual a ordem pertence (tipo caracter).
DBOI_FULLPATH	Nome do arquivo de índice (com seu diretório) ao qual a ordem pertence (tipo caracter)
DBOI_ORDERCOUNT	Número de ordens existentes no arquivo de índice especificado

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBORDERINFO para obter informações
sobre o nome do arquivo de índice da ordem corrente.
DBORDERINFO(DBOI_BAGNAME) // retorna: Ind
DBORDERINFO(DBOI_FULLPATH) // retorna: C:\AP6\Teste\Ind.cdx
// Este exemplo demonstra como se pode utilizar o DBORDERINFO para obter informações
sobre o nome do arquivo de índice de uma ordem especificada.
DBORDERINFO(DBOI_BAGNAME,,3) // retorna: Ind2
DBORDERINFO(DBOI_FULLPATH,,"Nome") // retorna: C:\AP6\Teste\Ind2.cdx
// Este exemplo demonstra como se pode utilizar o DBORDERINFO para obter o número de
ordens de determinado arquivo de índice.
DBORDERINFO(DBOI_ORDERCOUNT,"\Teste\Ind2.cdx") // retorna: 3
```

DBOrderNickName

Torna ativa a ordem com o determinado apelido.

Sintaxe

DBOrderNickName(*cApelido*) -> Lógico

Argumento	Obrigat.	Tipo	Descrição
cApelido	S	C	Nome do apelido da ordem a ser setada.

Retorno	Descrição
.F.	Não conseguiu tornar a ordem ativa. Principais erros: Não existe tabela ativa ou não foi encontrada a ordem com o apelido.
.T.	A ordem foi setada com sucesso.

Descrição

Esta função é utilizada para selecionar a ordem ativa através de seu apelido. Esta ordem é a responsável sequência lógica dos registros da tabela corrente.

Exemplo

```
//Este exemplo demonstra como se pode utilizar o DBORDERNICKNAME para setar nova ordem.
USE Cliente NEW
SET INDEX TO Nome, Idade
```

```
IF !DBORDERNI CKNAME ("IndNome")
    Messagebox("Registro não encontrado","Erro", 0)
    BREAK
ENDIF
```

DBRecall

Altera o estado deletado do registro atual.

Sintaxe

DBRecall() -> Nil

Retorno	Descrição
NIL	Nenhum

Descrição

Esta função 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.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBRECALL para retornar o estado do
registro atual para normal.
USE Cliente
DBGOTO(100)
DBDELETE()
DELETED() // Retorna: .T.
DBRECALL()
DELETED() // Retorna: .F.
// Este exemplo demonstra como se pode utilizar o DBRECALL para desfazer todas as
deleções da tabela corrente.
USE Cliente
DBGOTOP()
WHILE !EOF()
    DBRECALL()
    DBSKIP()
ENDDO
```

DBRecordInfo

Verifica informações sobre um registro.

Sintaxe

DBRecordInfo (*nInfoTipo* , [*nRegistro*]) -> Informação do Registro

Argumento	Obrigat.	Tipo	Descrição
nInfoTipo	S	N	Tipo de informação a ser verificada.
nRegistro	Opcional	N	Número do registro a ser verificado.

Retorno	Descrição
Nil	Não há tabela corrente ou registro inválido. Informação do Registro. Informação requisitada pelo usuário (o tipo depende da informação requisitada).

Descrição

Esta função é utilizada para obter informações sobre o registro especificado pelo segundo argumento (recono) 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:

DBRI_DELETED	Estado de deletado. Similar a DELETED (tipo lógico)
DBRI_RECSize	Tamanho do registro. Similar a RECSIZE (tipo numérico)
DBRI_UPDATED	Verifica se o registro foi alterado e ainda não foi atualizado fisicamente. Similar a UPDATED (tipo lógico).

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBRECORDINFO para se obter as
informações sobre registros da tabela corrente.
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

Reconstrói todos os índices da área de trabalho.

Sintaxe

DBReindex() -> Nil

Retorno	Descrição
NIL	Nenhum

Descrição

Reconstrói todos os índices da área de trabalho corrente e posiciona as tabelas no primeiro registro lógico.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBREINDEX para reconstruir os índices  
depois que um novo índice foi gerado.  
USE Clientes NEW  
DBSETINDEX( "IndNome" )  
DBREINDEX()
```

DBRLOCK

Bloqueia determinado registro.

Sintaxe

DBRLOCK([nRegistro]) --> Lógico

Argumento	Obrigat.	Tipo	Descrição
nRegistro	Opcional	N	Número do registro a ser bloqueado.

Retorno	Descrição
.F.	Não conseguiu bloquear o registro. Principal motivo: o registro já foi bloqueado por outro usuário.
.T.	O registro foi bloqueado com sucesso

Descrição

Esta função 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.

Exemplo

```
// Este exemplo mostra duas variações do uso de DBRLOCK.
DBUSEAREA( .T., "dbfcdxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F. )
DBGOTO(100)
DBRLOCK() // Bloqueia o registro atual (100)
DBRLOCK(110) // Bloqueia o registro de número 110
```

DBRLOCKLIST

Retorna uma lista com todos os registros locados na tabela corrente.

Sintaxe

DBRLOCKLIST() --> Array

Retorno	Descrição
Nil	<i>Não existe tabela corrente ou não existe nenhum registro locado.</i>
ListaLock	Lista com os reynos dos registros locados na tabela corrente.

Descrição

Esta função é utilizada para verificar quais registros estão locados na tabela corrente. Para tanto, é retornada uma tabela unidimensional com os números dos registros.

Exemplo

```
// Este exemplo mostra como é utilizada a função DBRLOCKLIST para verificar quais
registros estão bloqueados na tabela corrente
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

Desbloqueia determinado registro.

Sintaxe

DBRUNLOCK([*nRegistro*]) --> Nil

Argumento	Obrigat.	Tipo	Descrição
nRegistro	Não	N	Número do registro a ser desbloqueado.

Retorno	Descrição
NIL	Sem retorno.

Descrição

Esta função 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.

Exemplo

```
// Este exemplo mostra duas variações do uso de DBRUNLOCK.  
DBUSEAREA( .T., "dbfcdxads", "\dadosadv609\sai1990.dbf", "SSS", .T., .F. )  
DBGOTO(100)  
DBRUNLOCK() // Desbloqueia o registro atual (100)  
DBRUNLOCK(110) // Desbloqueia o registro de número 110
```

DbSeek

Encontra um registro com determinado valor da chave do índice.

Sintaxe

DBSeek (*cExp* | *nExp*, [*ISoftSeek*], [*IUltimo*]) --> Lógico

Argumento	Obrigat.	Tipo	Descrição
<i>cExp</i>	Sim	C	Valor de chave a ser encontrado do tipo caracter (todos os tipos de expressão de índice com exceção do índice com apenas um campo do tipo numérico).
<i>nExp</i>	Sim	N	Valor de chave a ser encontrado do tipo numérico (apenas quando a expressão do índice possui apenas um campo do tipo numérico).

ISoftSeek	Não	L	Posiciona no primeiro registro com expressão de chave maior que o valor procurado. O padrão é .F.
IUltimo	Não	L	Procura a última ocorrência do valor procurado. O padrão é .F.

Retorno	Descrição
.F.	Não foi encontrado nenhum registro com o valor especificado
.T.	Foi encontrado um registro com o valor especificado

Descrição

Esta função é utilizada para encontrar um registro com determinado valor da expressão de chave de índice.

Antes da chamada do DBSEEK deve-se certificar de que existe uma ordem ativa no momento com os campos que se deseja pesquisar o valor.

Se a expressão possuir apenas um campo numérico, o primeiro parâmetro deve ser do tipo numérico, mas nos demais casos deve-se utilizar um valor do tipo caracter para este parâmetro (mesmo se forem apenas dois campos numéricos ou do tipo data).

Quando o segundo parâmetro for especificado como .T. (softseek), mesmo que a expressão pesquisada não encontrar nenhum registro com este valor, a tabela será posicionada no próximo valor maior que o especificado no primeiro parâmetro, mas mesmo posicionando no próximo valor esta função retornará .F. (pois não encontrou).

Quando não for especificado este valor ou estiver .F. e falhar o valor de pesquisa, a tabela será posicionada em LASTREC + 1 e será setada a flag de EOF.

Se o terceiro parâmetro for especificado com valor .T. a função posiciona a tabela no último registro com o valor procurado, caso não seja especificado ou for .F., será posicionada na primeira ocorrência.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBSEEK para busca de valores numéricos.
USE Clientes NEW
ORDLISTADD ("/teste/ind1.cdx") // Expressão é Num (campo numérico)
DBSEEK(100) // Retorna: .F.
EOF() // Retorna: .T.
DBSEEK(100,.T.) // Retorna: .F.
EOF() // Retorna: .F. (pois o softseek posicionou no próximo registro)
// Este exemplo demonstra como se pode utilizar o DBSEEK para percorrer todos os
registros de Clientes com o nome joao e vencimentos a partir de janeiro de 2001.
USE Clientes NEW
ORDLISTADD ("/teste/ind2.cdx") // Expressão é Nome+Venc (campo caracter + data)
DBSEEK("joao200101",.T.) // Procura a primeira ocorrência de Nome "joao" e vencimento
maior que Janeiro de 2001
```

```
WHILE !EOF() .AND. Nome == "      joao"
  DBSKIP()
ENDDO
```

DBSetDriver

Modifica ou verifica o RDD padrão.

Sintaxe

DBSetDriver ([*cNovoRddPadrão*]) --> Caracter

Argumento	Obrigat.	Tipo	Descrição
cNovoRddPadrão	Não	C	Novo nome do RDD a ser definido como padrão.

Retorno	Descrição
cAntigoRddPadrão	Nome do RDD padrão corrente

Descrição

Esta função 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.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBSETDRIVER para alterar o valor do RDD padrão.
DBSETDRIVER("CTREECDX") // Retorna: DBFCDX
DBSETDRIVER() // Retorna: CTREECDX
```

DBSetFilter

Seta uma condição de filtro.

Sintaxe

DBSetFilter(*bCondição*, *cCondição*) --> Nil

Argumento	Obrigat.	Tipo	Descrição
bCondição	Sim	Bloco de Código	Expressão do filtro na forma executável.
cCondição	Sim	C	Expressão do filtro na forma de string.

Retorno	3Descrição
NIL	Sem retorno.

Descrição

Esta função é utilizada para setar um filtro nos registros da tabela corrente especificado através do bloco de código no primeiro parâmetro.

Quando um registro não está dentro do filtro setado ele continua existindo fisicamente, mas não logicamente (nas funções de manipulação de banco de dados como [DBGOTOP](#), [DBSEEK](#), [DBSKIP](#), etc).

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBSETFILTER para filtrar todos os
// clientes com menos de 40 anos.
USE Cliente NEW
DBSETFILTER( {|| Idade < 40}, "Idade < 40" )
DBGOTOP()
```

DBSetIndex

Acrescenta todas as ordens de determinado índice à lista.

Sintaxe

DBSetIndex(*cArqIndice*) --> Nil

Argumento	Obrigat.	Tipo	Descrição
<i>cArqIndice</i>	Sim	C	Nome do arquivo de índice, com ou sem diretório e extensão.

Retorno	Descrição
---------	-----------

NIL

Sem retorno.

Descrição

Esta função 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.

Para se utilizar arquivos de extensão padrão do RDD, este dado pode ser omitido no primeiro parâmetro, mas caso contrário deve ser especificado.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBSETINDEX para acrescentar novos índices à lista de ordens.  
USE Cliente NEW  
DBSETINDEX("Ind1")  
DBSETINDEX("\teste\Ind2.cdx")
```

DBSetNickName

Seta um apelido para determinada ordem ou verifica qual é o apelido corrente.

Sintaxe

DBSetNickName(*cOrdem*, [*cApelido*]) --> Caracter

Argumento	Obrigat.	Tipo	Descrição
<i>cIndice</i>	Sim	C	Nome da ordem que deve receber o apelido.
<i>cApelido</i>	Não	C	Nome do apelido da ordem a ser setada.

Retorno	Descrição
""	Não conseguiu encontrar a ordem especificada, não conseguiu setar o apelido ou não havia apelido corrente
<i>cApelido</i>	Apelido corrente

Descrição

Esta função 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.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBSETNICKNAME para setar um novo
apelido e verificar qual o apelido atual.
USE Cliente NEW
DBSETNICKNAME("IndNome") // retorna: ""
DBSETNICKNAME("IndNome", "NOME") // retorna: ""
DBSETNICKNAME("IndNome") // retorna: "NOME"
```

DBSetOrder

Seleciona a ordem ativa da área de trabalho.

Sintaxe

DBSetOrder(*nPosição*) --> Nil

Argumento	Obrigat.	Tipo	Descrição
<i>nPosição</i>	Sim	N	Posição da ordem na lista de ordens ativas

Retorno	Descrição
NIL	Sem retorno

Descrição

Esta função é utilizada para selecionar a ordem ativa da área de trabalho.

Esta ordem é a responsável sequência lógica dos registros da tabela corrente.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBSETORDER para selecionar a ordem
corrente.
```

```
USE Cliente NEW
SET INDEX TO Nome, Idade
DBSETORDER(2)
```

DBSkip

Desloca para outro registro na tabela corrente.

Sintaxe

DBSkip([*nRegistros*]) --> Nil

Argumento	Obrigat.	Tipo	Descrição
nRegistros	Não	N	Número de registros a ser deslocado.

Retorno	Descrição
NIL	Sem retorno.

Descrição

Esta função é utilizada para deslocar para outro registro a partir do registro atual.

O parâmetro especifica quantos registros lógicos devem ser deslocados a partir do corrente, se for positivo desloca em direção ao final da tabela, se for negativo ao início da tabela e caso seja omitido irá para o próximo registro (o padrão é 1).

Caso passe do início da tabela, posiciona no primeiro registro e seta BOF, caso passe do final da tabela, posiciona no registro LASTREC + 1 e seta EOF.

Exemplo

```
// Este exemplo mostra como o DBSKIP pode passar do final da tabela e do início da tabela
DBUSEAREA( .T., "dbfcdxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F. )
DBGOBOTTOM()
EOF() // retorna .F.
DBSKIP()
EOF() // retorna .T.
DBGOTOP()
BOF() // retorna .F.
DBSKIP(-1)
BOF() // retorna .T.
// Este exemplo mostra como o DBSKIP pode deslocar 10 registro em relação ao registro
corrente
DBUSEAREA( .T., "dbfcdxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F. )
DBGOTO(100)
DBSKIP(10)
RECNO() // retorna 110
```

```
DBSKIP(-10)
RECNO() // retorna 100
```

DBStruct

Retorna a estrutura da tabela corrente.

Sintaxe

DBStruct() --> Array

Retorno	Descrição
{}	<i>Não existe tabela corrente.</i>
aEstrutura	Array com todos os campos, onde cada elemento é um subarray contendo Nome, Tipo, Tamanho e Decimais.

Descrição

Esta função é utilizada para verificar a estrutura da tabela corrente da mesma forma que é utilizada para criar a tabela com a função [DBCCREATE](#).

Para isto ela cria um array para gravar as informações e as retorna.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBSTRUCT para recuperar a estrutura da
tabela corrente.
USE Cliente NEW
DBSTRUCT()
//Retorna: { {Cod,N,3,0}, {Nome,C,10,0}, {Idade,N,3,0}, {Nasc,D,8,0}, {Pagto,N,7,2} }
```

DBUnlock

Desbloqueia todos os registros da tabela corrente.

Sintaxe

DBUNLOCK() --> Nil

Retorno	Descrição
---------	-----------

NIL	Sem retorno.
-----	--------------

Descrição

Esta função é utilizada para liberar todos os registros bloqueados e é equivalente a executar [DBRUNLOCK](#) para todos os registros da lista [DBRLOCKLIST](#).

Exemplo

```
// Este exemplo mostra como liberar todos os registros bloqueados da tabela corrente.  
DBUSEAREA( .T., "dbfcdxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F. )  
DBUNLOCK()
```

DBUnlockAll

Desbloqueia todos os registros de todas as tabelas abertas.

Sintaxe

DBUNLOCKALL() --> Nil

Retorno	Descrição
NIL	Sem retorno.

Descrição

Esta função é utilizada para liberar todos os registros bloqueados e é equivalente a executar [DBUNLOCK](#) para todos os registros da lista [DBRLOCKLIST](#) de todas as áreas de trabalho.

Exemplo

```
// Este exemplo mostra como liberar todos os registros bloqueados da tabela corrente.  
DBUSEAREA( .T., "dbfcdxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F. )  
DBUNLOCKALL()
```

Delete

Marca registros como deletados.

Sintaxe

DELETE [FOR *CondFor*] [WHILE *CondWhile*] [NEXT *nRecs*] [RECORD *nRecno*] [REST] [ALL]

Argumento	Obrigat.	Tipo	Descrição
CondFor	Não		Expressão em ADVPL a ser resolvida para que o registro seja marcado como deletado.
CondWhile	Não		Expressão em ADVPL que determina quando a "deleção" deve parar (quando a expressão retornar .F.).
nRecs	Não	N	Quando registros devem ser marcados como deletados.
nRecno	Não	N	Número do recno do registro a ser deletado.

Retorno	Descrição
NIL	Sem retorno.

Descrição

Este comando é utilizado para marcar alguns registros da tabela corrente como deletados.

Se não forem especificadas as condições para que o registro seja marcado como deletado, marca toda a tabela como se estivesse especificada a opção "ALL".

Pode-se especificar um escopo para que os registros sejam marcados através das opções "FOR *CondFor*" e "WHILE *CondWhile*".

Pode-se também limitar o número de registros a serem marcados através da opção "NEXT *nRecs*" e determinar que a "deleção" dos registros deve-se iniciar a partir do registro atual com "REST", mas caso contrário o comando executa um [DBGOTOP\(\)](#) antes de iniciar a "deleção".

Se é desejado marcar apenas determinado registro pode-se defini-lo através da especificação do recno com "RECORD *nRecno*".

Exemplo

```
// Este exemplo demonstra como utilizar o comando DELETE para marcar alguns registros dentro de determinado escopo como deletado. Este escopo é definido por Idade > 20, até que o nome seja maior ou igual a "VVV", começa a deleção a partir do registro atual e marca apenas 10 registros:  
USE Cliente VIA "CTREECDX" NEW  
DELETE FOR Idade>20 WHILE Nome<"VVV" NEXT 10 REST
```

Deleted

Verifica se o registro foi deletado.

Sintaxe

DELETED() --> Lógico

Retorno	Descrição
.F.	O registro não foi deletado
.T.	O registro foi deletado

Descrição

Quando o registro é deletado, permanece fisicamente na tabela, mas fica marcado como "deletado". Esta função verifica este estado. Quando é executada a função DBPACK todos os registros marcados como deletados são apagados fisicamente enquanto a função [DBRECALL](#) marca todos os registros deletados como não-deletados.

Exemplo

// Este exemplo verifica se determinado registro está deletado, caso positivo, mostra uma mensagem:

```
USE "DADOSADV\AA1990.DBF" SHARED NEW
DBGOTO(100)
IF DELETED()
    MessageBox("O registro atual foi deletado","Erro", 0)
    BREAK
ENDIF
```

DevOutPict

Imprime o conteúdo na posição corrente formatando a saída.

Sintaxe

DEVOUTPICT (xVal, cPicture) --> NIL

Argumento	Obrigat.	Tipo	Descrição
xVal	Sim	C, N ou D	Informar valor a ser impresso.

cPicture	Sim	C	Informar a máscara de formatação do dado.
----------	-----	---	---

Retorno	Descrição
NIL	Nenhum

Descrição

DEVOUTPICT() imprime conteúdo na posição corrente de [PROW\(\)](#) e [PCOL\(\)](#), aplicando a máscara cPicture.

Exemplo

```
nVal:= 100.89
DevPos( 10,10 ) // configura linha 0, coluna 0 para impressão
//@ Prow(),Pcol() PSAY "Minha linha e "+str( Prow() )
DevOutPict( nVal, "@E 999,999.99" ) // imprime 100,89
```

DevPos

Posiciona linha e coluna de impressão.

Sintaxe

DEVPOS(nLin, nCol) --> NIL

Argumento	Obrigat.	Tipo	Descrição
nLin	Sim	N	Informar nova linha de impressão.
nCol	Sim	N	Informar nova coluna de impressão.

Retorno	Descrição
NIL	Nenhum

Descrição

DEVPOS() modifica a linha e coluna corrente de impressão. Modifica os retornos de [PROW\(\)](#) e [PCOL\(\)](#).

Exemplo

```
DevPos( 10,10 ) // configura linha 0, coluna 0 para impressao
@ Prow(),Pcol() PSAY "Minha linha e "+str( Prow() )
```

Directory

Cria um array com dados dos diretórios e dos arquivos.

Sintaxe

DIRECTORY(*cDirSpec* , [*cAtributos*]) -->Array

Argumento	Obrigat.	Tipo	Descrição	
cDirSpec	Sim	C	Identifica o drive, o caminho de onde as informações serão buscadas. O Caminho pode estar no servidor ou no local (Remote).	
cAtributos	Não	C	Especifica os atributos especiais de arquivos/diretórios que devem ser incluídos na busca por informações. Abaixo estão os atributos aceitos. Note que o atributo normal é sempre considerado, desde que não seja informado.	
			Atributo	Descrição
			H	Incluir arquivos ocultos (Hidden)
			S	Incluir arquivos de sistema.
			D	Incluir diretórios
			V	Procurar o nome do disco (label)

Retorno	Descrição	
Array	Retorna um array de subarrays onde cada subarray contém informações sobre um arquivo ou diretório que correspondem a <i>cDirSpec</i> . Os subarrays possuem a seguinte estrutura, definida em <i>directry.ch</i> :	
	F_NAME	Caracter, Nome do arquivo
	F_SIZE	Numérico, Tamanho do arquivo
	F_DATE	Data, Data de modificação do arquivo
	F_TIME	Caracter, Hora de modificação do arquivo
	F_ATTR	Caracter, Atributos do arquivo.

Descrição

DIRECTORY() é função que retorna informações de arquivos/diretórios que correspondam a uma máscara específica.

Tanto arquivos no servidor como no cliente (Remote) são aceitos. Os atributos são sempre aditivos.

Utilize Directory() ao invés de ADIR().

Exemplos

```
#include "Directry.ch"

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

DirRemove

Elimina um diretório.

Sintaxe

DIRREMOVE(*cDirNome*) --> ISuccesso

Argumento	Obrigat.	Tipo	Descrição
<i>cDirNome</i>	Sim	C	Nome do diretório a ser removido, opcionalmente incluindo o caminho. O diretório será removido no servidor.

Retorno	Descrição
.T.	O diretório foi eliminado.
.F.	Não foi possível eliminar o diretório.

Descrição

DIRREMOVE() elimina um diretório específico.

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

Exemplo

```
DI RREMOVE("c:\teste\um\dois") // Eliminando um diretório, sem testar o retorno
```

```
bResult := DI RREMOVE("c:\teste\um") // Elimina e testa o retorno
IF bResult
    ? "Impossível excluir o diretório"
ENDIF
```

DiskSpace

Retorna o espaço disponível em um disco específico.

Sintaxe

DISKSPACE([*nDrive*]) --> *nBytes*

Argumento	Obrigat.	Tipo	Descrição
<i>nDrive</i>	Não	N	Número do drive, onde 0 é o espaço do diretório corrente, e 1 é o A: do cliente, B: é o 2 do cliente, etc.

Retorno	Descrição
<i>nBytes</i>	Número de bytes disponíveis no disco informado.

Descrição

DISKSPACE() é uma função de ambiente que determina quantos bytes estão disponíveis em um determinado disco.

Ao ser utilizado zero e o diretório corrente for local, será retornado o espaço disponível no cliente (remote), caso o diretório corrente esteja no servidor, será retornado o espaço disponível no servidor.

Exemplos

```
CURDIR( "\SIGAADV" ) // Posiciona no diretorio SIGAADV do servidor
? DI SKSPACE( 0 ) // Retorna o espaço disponível no servidor.
? DI SKSPACE( 1 ) // Retorna o espaço disponível no drive a: local ( remote ).
CURDIR( "C:\WINDOWS" ) // Posiciona no diretorio SIGAADV do servidor
? DI SKSPACE( 0 ) // Retorna o espaço disponível na estação.
```

Dow

Converte uma data para o valor numérico do dia da semana.

Sintaxe

DOW(dData) --> Numérico

Argumento	Obrigat.	Tipo	Descrição
dData	Sim	D	É o valor da data a converter.

Retorno	Descrição
>=0 e <=7	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.

Descrição

DOW() é uma 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.

Exemplos

Estes exemplos mostram como usar CDOW() e o seu relacionamento com DOW():

```
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
```

Esta função de usuário utiliza DOW() para calcular a data da última segunda-feira da data informada.

```
FUNCTION LastMonday(dData)
RETURN (dData - DOW(dData) + 2)
```

Dtoc

Converte uma data para cadeia de caracteres.

Sintaxe

DTOC(dData) --> Caracter

Argumento	Obrigat.	Tipo	Descrição
DData	Sim	D	É o valor a converter.

Retorno	Descrição
Caracter	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.

Descrição

DTOC() converte 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() <a>.

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

Exemplos

```
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

Converte uma data para uma cadeia de caracteres no formato yyyymmdd.

Sintaxe

DTOS(*dData*) --> Caracter

Argumento	Obrigat.	Tipo	Descrição
<i>dData</i>	Sim	D	É o valor da data a converter.

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

Descrição

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

Exemplos

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

Este exemplo mostra como é criado um índice composto usando DTOS():

```
USE Vendas NEW
INDEX ON DTOS(Data) + Vendedor TO DataNome
```

Eject

Força a impressão de nova página no relatório.

Sintaxe

```
EJECT( )--> NIL
```

Retorno	Descrição
NIL	Sem retorno

Descrição

EJECT() efetua o salto de página na impressão de relatório configurando o [PROW\(\)](#) para 0.

Exemplo

```
SETPRC(0,0) // inicia impressao na linha 0 coluna 0
PCOL( 10 ) //muda para coluna 10
@ ROW( ), PCOL( ) PSAY "Minha Linha nesta pagina e " + str( PROW( ) )
PRNFLUSH( ) // Forca impressao antes do termino da pagina.
PROW( PROW( )+2 ) // Pula 2 linhas
PCOL( 10 )
@ PROW( ), PCOL( ) PSAY "Minha Linha nesta pagina e " + str( PROW( ) )
//PROW( 0 ) // Salta a pagina
EJECT( ) // Salta de pagina
FECHAREL( ) // Fecha impressao
```

ElapTime

Retorna o tempo decorrido entre duas horas.

Sintaxe

```
ElapTime( cHoraInicial , cHoraFinal ) --> Caracter
```

Argumento	Obrigat.	Tipo	Descrição
cHoraInicial	Sim	C	Informe a hora inicial no formato hh:mm:ss, onde hh é a hora (1 a 24), mm os minutos e ss os segundos
cHoraFinal	Sim	C	Informe a hora final no formato hh:mm:ss, onde hh é a hora (1 a 24), mm os minutos e ss os segundos.

Retorno	Descrição
Caracter	A diferença de tempo no formato hh:mm:ss, onde hh é a hora (1 a 24), mm os minutos e ss os segundos

Descrição

ElapTime() 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.

Exemplos

Este exemplo compara ELAPTIME():

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

FClose

Fecha um arquivo binário e grava os buffers no disco.

Sintaxe

FCLOSE(nHandle) --> Lógico

Argumento	Obrigat.	Tipo	Descrição
nHandle	Sim	N	Handle a ser fechado

Retorno	Descrição
---------	-----------

.F.	nHandle informado não corresponde a um arquivo aberto previamente, ou erro na gravação dos buffers pendentes, verifique através de FERROR() maiores detalhes sobre o erro.
.T.	Arquivo foi fechado com sucesso.

Descrição

Fecha um manipulador (handle) de um arquivo binário. Este manipulador deve ter sido aberto através das funções [FOPEN\(\)](#), [FOPENPORT\(\)](#), [FCREATE\(\)](#).

Exemplo

O exemplo abaixo utiliza o CLOSE para fechar um arquivo.

```
#include "Fileio.ch"
//
nHandle := FCREATE("Testfile", FC_NORMAL)
IF !FCLOSE(nHandle)
    conout ("Erro fechando arquivo:" + Str(FError( 0 ))
ENDIF
```

FCreate

Cria um arquivo, ou elimina o conteúdo de um arquivo.

Sintaxe

FCREATE(cArq, [nAtributo]) --> nHandle

Argumento	Obrigat.	Tipo	Descrição
cArq	S	C	Nome do arquivo a ser criado, o caminho pode ser incluído. São válidos caminhos no local (remote) e no servidor.
nAtributo	Opcional	N	Atributos no qual o arquivo será criado.

Retorno	Descrição
>0	Numérico. O Handle do arquivo para ser usado nas demais funções de arquivo.
-1	Erro na criação do arquivo, verifique FERROR() para obter maiores detalhes do erro.

Descrição

FCREATE() é uma 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.

O Arquivo deve ser fechado através da função `FCLOSE()`.

Os Atributos para criação do arquivo estão abaixo, e estão definidos no header `fileio.ch`.

FC_NORMAL	Criação normal do Arquivo (default/padrão).
FC_READONLY	Cria o arquivo protegido para gravação.
FC_HIDDEN	Cria o arquivo como oculto.
FC_SYSTEM	Cria o arquivo como sistema.

Note que se o arquivo existir, o conteúdo será truncado para zero bytes.

Exemplo

```
#include "Fileio.ch"
IF (nHandle := FCREATE("Testfile", FC_NORMAL)) == -1
? "Arquivo não pode ser criado", FERROR()
BREAK
ELSE
FWRITE(nHandle, "ola!!!")
FCLOSE(nHandle)
ENDIF
```

FErase

Apaga um arquivo do disco.

Sintaxe

`FERASE(cArg) --> nSucesso`

Argumento	Obrigat.	Tipo	Descrição
cArg	S	C	Apaga um arquivo do disco, aceita caminhos locais (Remote), ou caminhos no servidor Advanced Protheus.

Retorno	Descrição
	Arquivo foi apagado com sucesso.
-1	O Arquivo não foi apagado, verifique FERROR() para obter maiores detalhes.

Descrição

Elimina um arquivo. O Arquivo pode estar no servidor ou no local (Remote).

O Arquivo para ser apagado deve estar fechado. Não é permitido a utilização de caracteres coringa (wildcards).

Exemplos

```
// Este exemplo apaga todos os arquivos .BAK do diretório corrente.
#include "Directry.ch"
AEVAL(DIRECTORY("*.BAK"), { |aFile| FERASE(aFile[F_NAME]) })
/// Este exemplo apaga um arquivo no cliente ( Remote ) e imprime uma mensagem caso //
ele apresente erro.
IF FERASE("AFile.txt") == -1
? "File erase error:", FERROR()
BREAK
ENDIF
```

FError

Verifica se houve erros após uma operação com arquivos binários.

Sintaxe

FERROR() --> Numérico

Retorno	Descrição
0	Operação realizada com sucesso. Sem erros
2	Arquivo não encontrado
3	Caminho não encontrado
4	Muitos arquivos abertos
5	Acesso negado
6	Manipulador Invalido (Handle)
8	Memória Insuficiente

15	Drive inválido especificado
19	Tentativa de gravar em disco protegido contra-gravação
21	Drive Não esta pronto
23	Dados com erro de CRC
29	Erro de gravação
30	Erro de leitura
32	Violação de compartilhamento
33	Erro de Lock

Descrição

FERROR() é a função que retorna o código de erro mais específico, após a execução de funções com arquivos binários, tais como: [FOPEN\(\)](#), [FCLOSE\(\)](#), [FWRITE\(\)](#), [FREAD\(\)](#).

Se FERROR() retornar zero, indica sucesso na última operação.

FERROR() sempre retorna o estado da ultima operação.

Exemplo

```
#include "Fileio.ch"

//
nHandle := FCREATE("Temp.txt", FC_NORMAL)
IF FERROR() != 0
    conout ("Erro ao criar o arquivo, Erro:" Str(FError()))
ENDIF
```

FieldBlock

Retorna um bloco de código para um campo determinado da tabela corrente.

Sintaxe

FIELDBLOCK(*cCampo*) --> Bloco de Código

Argumento	Obrigat.	Tipo	Descrição
cCampo	S	C	Nome do campo a ser retornado o bloco de código.

Retorno	Descrição
---------	-----------

bBloco	Bloco de código. Bloco de código para o campo especificado na tabela corrente.
--------	--

Descrição

Esta função é utilizada para retornar um bloco de código executável com o campo especificado. Sendo que quando o bloco resultante é executado sem valor recupera o valor armazenado no campo, mas quando executado com um valor, seta este valor no determinado campo.

Portanto, o bloco retornado é similar a: &("{ |Valor| IF(Valor==NIL, Campo, Campo: =Valor)}")

Sendo: Campo = parâmetro da função FIELDBLOCK()

Valor = valor executado no bloco de código

Exemplo

```
// Este exemplo mostra como se pode usar o FIELDBLOCK para criar o bloco de código para o
// campo Nome da tabela corrente na variável bBloco:
USE Cliente VIA "DBFCDX" NEW
bBloco := FIELDBLOCK("Nome")
```

FieldWbl

Retorna um bloco de código para um campo determinado especificando a área de trabalho.

Sintaxe

FIELDWBLOCK(<cCampo>, <nArea>) --> Bloco de Código

Argumento	Obrigat.	Tipo	Descrição
cCampo	S	C	Nome do campo a ser retornado o bloco de código.
cArea	S	N	Número da área de trabalho a ser criado o bloco de código.

Retorno	Descrição
bWBloco	Bloco de código. Bloco de código para o campo especificado na área de trabalho determinada.

Descrição

Esta função é utilizada para retornar um bloco de código executável com o campo especificado em determinada área de trabalho. Sendo que quando o bloco resultante é executado sem valor recupera o valor armazenado no campo, mas quando executado com um valor seta esta valor no determinado campo.

Portanto o bloco retornado é similar a:

```
&("{ |Valor| IF(Valor==NIL, nArea->Campo, nArea->Campo: =Valor)}")
```

Sendo: Campo = parâmetro da função FIELDBLOCK()

nArea = número da área de trabalho especificada no função FIELDWBLOCK()

Valor = valor executado no bloco de código

Exemplo

```
// Este exemplo mostra como se pode usar o FIELDWBLOCK para criar o bloco de código para
o campo Nome da área de trabalho 3 na variável bBloco:
USE Cliente VIA "DBFCDX" NEW
bBloco := FIELDBLOCK("Nome",3)
```

File

Verifica se um arquivo ou máscara de arquivos existem.

Sintaxe

```
FILE( cArquivo ) --> lExiste
```

Argumento	Obrigat.	Tipo	Descrição
cArquivo	S	C	Nome do arquivo podendo incluir caminhos. Caminhos locais (Remote) ou caminhos de servidor são aceitos.

Retorno	Descrição
.F.	O Arquivo ou padrão de busca existem.
.T.	O Arquivo ou padrão não existem.

Descrição

Verifica se existe um arquivo ou um padrão de arquivos, no diretório. Os caracteres * e ? são aceitos.

Caminhos de servidor e locais são aceitos.

Exemplos

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

FOpen

Abre um arquivo binário.

Sintaxe

FOPEN(*cArq* , *nModo*) --> Numérico

Argumento	Obrigat.	Tipo	Descrição
cArq	Sim	C	Nome do Arquivo a ser aberto em modo binário.
nModo	Não	N	Modo como o arquivo deverá ser aberto. O Modo de abertura é um modo composto entre modo de acesso e modo de compartilhamento. O modo padrão de abertura é zero, o que indica sem compartilhamento e somente leitura.

Retorno	Descrição
-1	Erro na abertura do arquivo. Verifique <code>FERROR()</code> para maiores detalhes do erro
>0	Handle do arquivo, a ser usado com as demais funções de arquivo

Descrição

FOPEN() abre um arquivo de modo binário, permitindo a leitura ou gravação através das funções [FREAD\(\)](#), [FWRITE\(\)](#).

Todo arquivo aberto através da FOPEN(), deve ser fechado através da [FCLOSE\(\)](#).

O modo de abertura é composto de modo de acesso e modo de compartilhamento, esses modos estão descritos em `fileio.ch`.

Modos de Acesso	
FO_READ	Apenas Leitura
FO_WRITE	Apenas Gravação
FO_READWRITE	Leitura e Gravação

Modos de Compartilhamento	
FO_COMPAT	Modo de compatibilidade (Padrão)
FO_EXCLUSIVE	Exclusivo
FO_DENYWRITE	Não permite que outros abram o arquivo para gravação
FO_DENYREAD	Não permite que outros abram o arquivo para leitura
FO_DENYNONE	permite leitura e gravação
FO_SHARED	Igual ao FO_DENYNONE

O modo de acesso em combinação (+) com o modo de compartilhamento determina a maneira como o arquivo será aberto e sua acessibilidade em um ambiente de rede.

Serão aceitos tantos caminhos baseados no servidor, como caminhos locais (remote).

Exemplo

Este exemplo usa o FOPEN() para abrir um arquivo para leitura/gravação de maneira compartilhada e mostra uma mensagem de erro caso a abertura falhe:

```
#include "Fileio.ch"
//
nHandle := FOPEN("Temp.txt", FO_READWRITE + FO_SHARED)
IF FERROR() != 0
? "Impossível abrir o arquivo, Erro : ", FERROR()
BREAK
ENDIF
```

FOpenPort

Abre uma porta paralela ou serial.

Sintaxe

FOPENPORT(cPorta , [cParm], [nModo]) --> Numérico

Argumento	Obrigat.	Tipo	Descrição
cArq	Sim	C	Nome do Dispositivo a ser usado.
cParm	Sim	C	Características do dispositivo, apenas válido para portas seriais. Os parâmetros são: Velocidade (BPS), Paridade, DataBits, StopBits, Timeout de leitura (Opcional).

nModo	Não	N	Modo como o dispositivo deverá ser aberto. O Modo de abertura é um modo composto entre modo de acesso e modo de compartilhamento. O modo padrão de abertura é zero, o que indica sem compartilhamento e somente leitura.
-------	-----	---	--

Retorno	Descrição
-1	Erro na abertura do arquivo. Verifique <code>FERROR()</code> para maiores detalhes do erro
>0	Handle do dispositivo, a ser usado com as demais funções de arquivo.

Descrição

`FOPENPORT()` abre um dispositivo, permitindo a leitura ou gravação através das funções [FREAD\(\)](#), [FWRITE\(\)](#).

Todo arquivo aberto através da `FOPENPORT()`, deve ser fechado através da função [FCLOSE\(\)](#).

O modo de abertura é composto de modo de acesso e modo de compartilhamento, esses modos estão descritos em `fileio.ch`.

Modos de Acesso	
FO_READ	Apenas Leitura
FO_WRITE	Apenas Gravação
FO_READWRITE	Leitura e Gravação
Modos de Compartilhamento	
FO_COMPAT	Modo de compatibilidade (Padrão)
FO_EXCLUSIVE	Exclusivo
FO_DENYWRITE	Não permite que outros abram o arquivo para gravação
FO_DENYREAD	Não permite que outros abram o arquivo para leitura
FO_DENYNONE	permite leitura e gravação.
FO_SHARED	Igual ao FO_DENYNONE

O modo de acesso em combinação (+) com o modo de compartilhamento determina a maneira como o arquivo será aberto e sua acessibilidade em um ambiente de rede.

O dispositivo será aberto no diretório corrente, ou seja, caso o diretório esteja posicionado no cliente, o dispositivo será aberto no cliente (Remote), caso contrário será aberto no servidor.

Os parâmetros das portas seriais servem para configurar a comunicação serial e podem ser os seguintes:

Velocidade (BaudRate)	9600 (Default), 19200, 38400, 57600, 115200.
Paridade	N (Default, Sem Paridade), E (Par), O (Impar).
DataBits	6, 7 ou 8 (Default) Bits.
StopBits	1 (default) ou 2 Bits.
Timeout de leitura	1000 (Default), Valores acima de zero. Tempo em milissegundos de espera por dados a serem lidos.

Exemplo

Este exemplo usa o FOPENPORT() para abrir uma porta de impressão para gravação e mostra uma mensagem de erro caso a abertura falhe:

```
#include "Fileio.ch"
//
nHandle := FOPENPORT("LPT1:", "", FO_WRITE)
IF FERROR() != 0
? "Impossível abrir a impressora LPT1, Erro : ", FERROR()
BREAK
ENDIF
Este exemplo abre uma porta serial e espera para obter uma leitura da porta.

#include "Fileio.ch"
//
CURDIR( "C:\TEMP" ) // Posicionando no diretório cliente ( Remote )
nHandle := FOPENPORT("COM1:", "9600,N,8,1,10000", FO_WRITE)
IF FERROR() != 0
? "Impossível abrir a impressora LPT1, Erro : ", FERROR()
BREAK
ENDIF
FREAD(
```

FRead

Lê caracteres binários de um arquivo.

Sintaxe

FREAD(nHandle , @cBuffer , nQtdBytes) --> Numérico

Argumento	Obrigat.	Tipo	Descrição
nHandle	Sim	N	É o manipulador retornado pelas funções FOPEN(), FCREATE(), FOPENPORT()

			FOPENPORT()
cBufferVar	Sim	C	É o nome de um buffer onde os dados lidos deverão ser armazenados. O tamanho desta variável deve ser maior ou igual ao tamanho informado em <i>nQtdBytes</i> . Esta variável deve ser sempre passada por referência. (@ antes do nome da variável)
nQtdBytes	Sim	N	Número máximo de bytes que devem ser lidos.

Retorno	Descrição
Numérico	Quantidades de bytes lidos. Caso a quantidade seja menor que a solicitada, indicará erro ou final de arquivo. Verifique <code>FERROR()</code> para obter maiores detalhes

Descrição

`FREAD()` lê a partir um arquivo aberto, através de [FCLOSE\(\)](#), [FCREATE\(\)](#), [FOPENPORT\(\)](#), os dados e armazena no buffer informado. `FREAD()` lê normalmente caracteres de controle (ASC 128, ASC 0, etc.).

`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 *nQtdBytes*.

O buffer passado para leitura deve ser sempre pré-alocado e passado como referência. Caso contrário, os dados não poderão ser retornados.

`FREAD()` lê a partir da posição atual do ponteiro, que pode ser ajustado pelo [FSEEK\(\)](#) ou por [FWRITE\(\)](#), [FREADSTR\(\)](#).

Exemplos

Este exemplo lê 128 bytes em um buffer.

```
#define F_BLOCK 128
//
cBuffer := SPACE(F_BLOCK)
nHandle := FOPEN("Temp.txt")
//
IF FERROR() != 0
? "Abertura de arquivos com erro: ", FERROR()
ELSE
IF FREAD(nHandle, @cBuffer, F_BLOCK) <> F_BLOCK
? "Erro lendo arquivo"
ENDIF
FCLOSE(nHandle)
ENDIF
```

FReadStr

Lê caracteres de um arquivo binário.

Sintaxe

`FREADSTR(nHandle , nQtdBytes) --> Caracter`

Argumento	Obrigat.	Tipo	Descrição
<i>nHandle</i>	Sim	N	É o manipulador retornado pelas funções <code>FOPEN()</code> , <code>FCREATE()</code> , <code>FOPENPORT()</code> .
<i>nQtdBytes</i>	Sim	N	Número máximo de bytes que devem ser lidos.

Retorno	Descrição
Retorno variável	Retorna uma string contendo os caracteres lidos.

Descrição

`FREADSTR()` lê de um arquivo aberto, através de [FCLOSE\(\)](#), [FCREATE\(\)](#), [FOPENPORT\(\)](#).

`FREAD()` lerá até o número de bytes informado em *nQtdBytes* ou até encontrar um `CHR(0)`. Caso aconteça algum erro ou o arquivo chegue ao final, `FREAD()` retornará uma string menor do que *nQtdBytes* e colocará o erro em `FERROR()`.

`FREADSTR()` lê a partir da posição atual do ponteiro, que pode ser ajustado pelo `FSEEK()` ou por `FWRITE()`, `FREAD()`.

Exemplos

Este exemplo lê os 16 caracteres de um arquivo e imprime o seu conteúdo.

```
#include "Fileio.ch"
//
nHandle := FOPEN("New.txt", FC_NORMAL)
IF FERROR() != 0
? "Erro abrindo o arquivo", FERROR()
ELSE
cString := FREADSTR(nHandle, 16)
? cString
FCLOSE(nHandle)
ENDIF
```

Frename

Muda o nome de um arquivo.

Sintaxe

`FRENAME(cNomeAntigo, cNovoNome) --> nSucesso`

Argumento	Obrigat.	Tipo	Descrição
cOldFile	S	C	Nome do arquivo será renomeado, aceita caminhos do servidor e caminhos do cliente.
cNewFile	S	C	Novo nome do arquivo, aceita também caminho do servidor, e caminho do cliente.

Retorno	Descrição
	A mudança foi executada com sucesso.
-1	Ocorreu algum erro na mudança de nome, verifique se os dois caminhos estão no mesmo ambiente. (Local/Local, Servidor/Servidor). Verifique <code>FERROR()</code> para detalhes do erro.

Descrição

`FRENAME()` renomeia um arquivo para outro nome, tanto no servidor como na estação. Ao renomear um arquivo não esqueça que este arquivo deverá estar fechado.

Exemplos

```
// Renomeando arquivos no cliente.
IF FRENAME("C:\TEMP\ArqAntigo.txt", "C:\TEMP\ArqNovo.txt") == -1
? "Erro ao renomear ", FERROR()
ENDIF
// Renomeando arquivos no Servidor.
IF FRENAME("\SIGAADV\ArqAntigo.txt", "\SIGAADV\ArqNovo.txt") == -1
? "Erro ao renomear ", FERROR()
ENDIF
```

FSeek

Posiciona o arquivo binário.

Sintaxe

`FSEEK(nHandle , nOffset , [nOrigem]) --> Numérico`

Argumento	Obrigat.	Tipo	Descrição
nHandle	Sim	N	Manipulador obtido através das funções <code>FOPEN</code> , <code>FCREATE</code> .
nOffset	Sim	N	Número de bytes que o ponteiro do arquivo deve ser movido a partir da origem informada em <code>nOrigem</code> . O número pode ser positivo ou

			negativo. O destino não precisa existir no arquivo.
nOrigem	Não	N	Indica a partir de qual posição do arquivo, o <i>nOffset</i> será considerado.

Retorno	Descrição
Numérico	A nova posição do arquivo.

Descrição

FSEEK() 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*:

FS_SET	Ajusta a partir do início do arquivo.
FS_RELATIVE	Ajuste relativo a posição atual do arquivo.
FS_END	Ajuste a partir do final do arquivo

Exemplo

```
// Exemplo calcula o tamanho do arquivo.
#include "Fileio.ch"
//
// Abre o arquivo apenas para leitura
IF (nHandle := FOPEN("Temp.txt")) >= 0
//
// Pega o tamanho do arquivo
nLength := FSEEK(nHandle, 0, FS_END)
//
FCLOSE(nHandle)
ELSE
? "Erro na abertura do arquivo", FERROR()
ENDIF
```

FWrite

Grava em um arquivo binário.

Sintaxe

FWRITE(*nHandle* , *cBuffer* , [*nQtdBytes*]) --> Numérico

Argumento	Obrigat.	Tipo	Descrição
-----------	----------	------	-----------

<i>nHandle</i>	Sim	N	É o manipulador retornado pelas funções <code>FOPEN()</code> , <code>FCREATE()</code> , <code>FOPENPORT()</code> .
<i>cBuffer</i>	Sim	C	É o nome de um buffer de onde os dados serão gravados. O tamanho desta variável deve ser maior ou igual ao tamanho informado em <i>nQtdBytes</i> (caso seja informado o tamanho).
<i>nQtdBytes</i>	Não	N	Quantidade de bytes a serem gravados.

Retorno	Descrição
Numérico	Quantidade de bytes efetivamente gravados. Se o retorno for menor que o <i>nQtdBytes</i> , houve um erro na gravação. Verifique <code>FERROR()</code> para mais detalhes.

Descrição

`FWRITE()` grava em um arquivo aberto, através de [FCLOSE\(\)](#), [FCREATE\(\)](#), [FOPENPORT\(\)](#), os dados do buffer informado. `FWRITE()` grava normalmente caracteres de controle (ASC 128, ASC 0, etc.).

`FWRITE()` gravará até o número de bytes informado em *nQtdBytes*; caso aconteça algum erro, retornará um número menor que o *nQtdBytes*.

`FWRITE()` grava a partir da posição atual do ponteiro, que pode ser ajustado pelo [FSEEK\(\)](#) ou por [FREAD\(\)](#), [FREADSTR\(\)](#).

Exemplos

Este exemplo copia um arquivo para outro.

```
#include "Fileio.ch"
#define F_BLOCK 512
//
cBuffer := SPACE(F_BLOCK)
nInfile := FOPEN("Temp.txt", FO_READ)
nOutfile := FCREATE("Newfile.txt", FC_NORMAL)
lDone := .F.
//
DO WHILE !lDone
  nBytesRead := FREAD(nInfile, @cBuffer, F_BLOCK)
  IF FWRITE(nOutfile, cBuffer, nBytesRead) < ;
    nBytesRead
    ? "Erro gravando: ", FERROR()
    lDone := .T.
  ELSE
    lDone := (nBytesRead == 0)
  ENDIF
ENDDO
//
FCLOSE(nInfile)
FCLOSE(nOutfile)
```

GetImpWindows

Retorna lista de impressoras disponíveis para impressão.

Sintaxe

GETIMPWINDOWS(*IServer*) --> Array

Argumento	Obrigat.	Tipo	Descrição
IServer	Sim	L	Informar .T. se desejar lista de impressoras do Server e .F. se desejar lista de impressoras do Remote.

Retorno	Descrição
Array	Array com nome das impressoras disponíveis.

Descrição

GETIMPWINDOWS() retorna uma lista de impressoras disponíveis no spool do Server ou Remote. Se o Server está em ambiente Unix, a GETIMPWINDOWS() retornará a lista com os nomes de impressoras cadastradas na chave PRINTERSNAME do AP6SRV.INI (ver [PREPAREPRINT\(\)](#)).

Exemplo

```
aImpressoras:= GetImpWindows(.F.) // retorna lista de impressoras do Windows do remote.
```

GetPortActive

Retorna lista de portas de impressão disponíveis.

Sintaxe

GETPORTACTIVE (*IServer*) --> Array

Argumento	Obrigat.	Tipo	Descrição
IServer	Sim	L	Informar .T. se desejar lista de impressoras do Server e .F. se desejar lista de impressoras do Remote.

Retorno	Descrição
---------	-----------

Array	Array com nome das impressoras disponíveis
-------	--

Descrição

GETPORTACTIVE() retorna uma lista de portas de impressão disponíveis do Server ou Remote. Se o Server está em ambiente Unix, a GETPORTACTIVE() retornará uma lista com os nomes de devices possíveis para Impressão.

Exemplo

```
aPortas:= GetPortActive(.F.) // retorna lista de portas de impressao do remote.
```

GetClientDir

Retorna o diretório onde está instalado o Remote.

Sintaxe

```
GetClientDir( ) --> cDir
```

Retorno	Descrição
cDir	Caracter, diretório aonde está instalado o remote.

Descrição

Retorna o diretório onde o Remote está instalado.

Exemplo

```
? GetClientDir() // Imprime o diretório onde está instalado o remote  
// Exemplo de saída  
// c:\ap6\bin\remote
```

Header

Verifica o tamanho do cabeçalho da tabela corrente.

Sintaxe

```
HEADER() --> Numérico
```

Retorno	Descrição
---------	-----------

0	Não há tabela corrente
nBytes	Tamanho do cabeçalho da tabela corrente em número de bytes

Descrição

Esta função pode ser utilizada em conjunto com as funções [RecSize](#) e [RecCount](#) para calcular o tamanho ocupado no disco pela tabela corrente. Pois, o tamanho será Header+RecSize*RecCount.

Exemplo

```
// Este exemplo calcula o tamanho ocupado pela tabela "AA1990.DBF" no disco (número de bytes):
USE "\DADOSADV\AA1990.DBF" SHARED NEW
nCabeçalho := HEADER()
nDados := RecSize() * RecCount()
nTamanhoTotal := nCabeçalho + nDados
```

IndexKey

Verifica qual a expressão de chave de um índice.

Sintaxe

INDEXKEY(*nOrdem*) --> Caracter

Argumento	Obrigat.	Tipo	Descrição
nOrdem	S	N	Posição do índice na lista.

Retorno	Descrição
""	Não existe índice para o parâmetro <i>nOrdem</i> corrente.
cExpOrdem	Expressão de chave da ordem ativa ou especificada pelos parâmetros.

Descrição

Esta função é utilizada para verificar qual é a expressão de chave de determinado índice, especificado pela posição do índice na lista pelo primeiro parâmetro.

Se for passado parâmetro igual a 0, será retornada a expressão de chave do índice atual.

Exemplo

```
// Este exemplo mostra como o INDEXKEY() pode recuperar a expressão do índice atual:
```



```

USE Cliente NEW
INDEX ON Nome+Cod TO Ind1 FOR Nome+Cod > "AZZZZZZZ"
INDEX ON Nome TO Ind2 FOR Nome > "CCCCCCC"
INDEXKEY(1) // Retorna: Nome+Cod
INDEXKEY(2) // Retorna: Nome
INDEXKEY(0) // Retorna: Nome - expressão corrente

```

IndexOrd

Verifica a posição do índice corrente.

Sintaxe

INDEXORD() --> Numérico

Retorno	Descrição
0	Não existe índice ou tabela corrente
nOrd	Posição do índice corrente na lista

Descrição

Esta função retorna a posição ocupada pelo índice corrente dentro da lista de índices.

Exemplo

```

// Este exemplo verifica qual a posição do índice corrente dentro da lista
USE Cliente NEW
SET INDEX TO Nome, End, Cep
INDEXORD() // Returns: 1 - é o primeiro índice da lista

```

InitPrint

Inicializa parâmetros de impressão de relatório.

Sintaxe

INITPRINT([nOutPut], [cNameRel], [cType], [lPort], [cPathAtu]) --> NIL

Argumento	Obrigat.	Tipo	Descrição
nOutPut	Não	N	Direcionamento da saída de impressão. 1=via Client (padrão) e 2=via Server
cNameRel	Não	C	Nome do relatório que será exibido no spool de impressão
cType	Não	C	Tipo do relatório. "220" = 220 colunas, "132L"= 132 colunas landscape (Padrão), "132P"= 132 colunas portrait, "080" = 80

			colunas portrait
IPort	Não	L	Não utilizado
cPathAtu	Não	L	Não utilizado

Retorno	Descrição
NIL	Sem retorno

Descrição

INITPRINT() inicializa parâmetros de impressão de relatório. Define onde será a saída de impressão e qual vai ser a formatação de saída.

Exemplo

```
/* Inicializa relatório para imprimir via Client "Meu relatório" em formulário de 132
colunas
em Landscape */
InitPrint( 1,"Meu relatório", "132L" )
```

IsPrinter

Verifica se impressora está disponível.

Sintaxe

ISPRINTER([xPorta], [ISeta], [nWhere], [@nErrorCode]) --> Lógico

Argumento	Obrigat.	Tipo	Descrição
xPorta	Não	C ou N	Informar a porta que desejamos verificar "LPT1" (padrão), "LPT2" , "COM1", "COM2" ou 1=LPT1, 2=LPT2.
ISeta	Não	L	Não utilizado
nWhere	Não	N	1= testa impressora no Remote (default), 2= testa impressora no Server.
nErrorCode	Não	N	Retorna o código de erro da impressora

Retorno	Descrição
.T.	Impressora ativa
.F.	Impressora inativa

Descrição

ISPRINTER() testa a disponibilidade de impressoras conectadas fisicamente à máquina onde está sendo executado o Server ou Remote Protheus. Dependendo do sistema operacional, o teste pode ser um Assembler em Windows 95 ou 98, ou a criação de um arquivo em Windows NT, 2000 e Unix.

Nos casos onde o teste é efetuado por criação de arquivo, o *nErrorCode* não será informado.

Exemplo

```
If !IsPrinter(1,,1,@nErr) // testa se impressora esta conectada na lpt1 no Remote.  
MsgBox("Impressora não conectada! Erro: "+Str(nErr))  
endif
```

IsCisaSyncOn

Verifica se o uso do CisaSync está habilitado.

Sintaxe

IsCisaSyncON() -> Lógico

Retorno	Descrição
.F.	Não está habilitado.
.T.	Está habilitado.

Descrição

Esta função é utilizada para verificar se o uso do CisaSync está habilitado ou não.

Exemplo

```
// Este exemplo demonstra como se pode utilizar IsCisaSyncOn verificar se o CisaSync está habilitado.  
USE Cliente NEW  
IF (ISCI SASYNCON())  
QOUT("CisaSync habilitado")  
ELSE  
QOUT("CisaSync desabilitado")  
ENDIF
```

Locate

Pesquisa um registro segundo um determinado escopo.

Sintaxe

LOCATE FOR *CondFor* [WHILE *CondWhile*] [NEXT *nRecs*] [RECORD *nRecno*] [REST] [ALL]

Argumento	Obrigat.	Tipo	Descrição
CondFor	S	Expressão AdvPL	Expressão em ADVPL a ser resolvida para o registro a ser aceito.
CondWhile	N	Expressão AdvPL	Expressão em ADVPL que determina quando a busca deve parar (quando a expressão retornar .F.)
nRecs	N	N	Quando registros devem ser verificados.
nRecno	N	N	Número do recno do registro a ser verificado.

Descrição

Este comando é utilizado para encontrar um registro com determinado valor dentro de um escopo especificado na tabela corrente.

Se a pesquisa obteve êxito, a flag FOUND retorna .T. e a tabela é posicionada no registro encontrado; caso FOUND retorne .F., a tabela é posicionada em EOF.

Se as opções do escopo forem omitidas (*CondWhile*, *nRecs* e *nRecno*), todos os registros são pesquisados, desde o primeiro como a opção "ALL".

Caso contrário, a opção "WHILE *CondWhile*" determina quando a busca deve parar (a expressão retornar .F.); "NEXT *nRecs*" determina quantos registros devem ser pesquisados na busca; "RECORD *nRecno*" determina o recno do registro a ser verificado na busca; "REST" significa que a busca deve começar a partir do próximo registro.

Exemplo

```
// Este exemplo demonstra como utilizar o comando LOCATE no seu modo mais usual:
USE Cliente NEW
LOCATE FOR Nome="Jose"
FOUND() // Retorna: .T.
EOF() // Retorna: .F.
RECNO() // Retorna: 5
LOCATE FOR Nome="DDDDD"
FOUND() // Retorna: .F.
EOF() // Retorna: .T.
RECNO() // Retorna: 85
// Este exemplo demonstra como se pode utilizar o comando LOCATE para lista todos os
// Cliente de nome "Jose" com idade 20 anos:
USE Cliente NEW
DBGOTOP()
LOCATE FOR Idade=20 WHILE Nome="Jose"
DO WHILE FOUND()
  DBSKIP()
  LOCATE REST FOR Idade=20 WHILE Nome="Jose"
ENDDO
```

LUpdate

Verifica a data da última modificação da tabela corrente.

Sintaxe

LUpdate() --> Data

Retorno	Descrição
Data em branco	Não há tabela corrente
Data	Data da última modificação da tabela corrente (dia, mês e ano)

Descrição

Esta função verifica qual a data da última modificação e fechamento da tabela corrente, caso não exista tabela corrente é retornada uma data em branco.

Exemplo

```
// Mostra a data da última modificação da tabela corrente, caso não exista tabela
corrente mostra uma mensagem de erro:
dModificacao := LUpdate()
IF (EMPTY(dModificacao))
    MessageBox("Não há tabela corrente","Erro", 0)
    BREAK
ELSE
    CONOUT ("A data da ultima modificacao e: " + DTOS(dModificacao))
ENDIF
```

MakeDir

Cria um diretório.

Sintaxe

MAKEDIR(*cNovoDir*) --> Numérico

Argumento	Obrigat.	Tipo	Descrição
<i>cNovoDir</i>	Sim	C	Nome do diretório a ser criado, incluindo opcionalmente o caminho.

Retorno	Descrição
---------	-----------

0	Diretório foi criado com sucesso
- 1	Erro na criação do diretório

Descrição

Cria um diretório na estação ou no servidor Advanced Protheus.

Exemplo

```
MAKEDI R("c:\teste\um") // Cria um diretório na estacao
nResult := MAKEDI R("\teste\um") // Cria o diretorio no servidor Advanced protheus
IF nResult != 0
    conout("Impossivel criar o diretório no servidor Protheus" + Str(nResult))
ENDIF
MAKEDI R( "teste" ) // Exemplo também válido ( Criando o diretório no servidor)
```

MemoLine

Extrai uma linha de uma string ou de um campo memo.

Sintaxe

MEMOLINE(cString, [nLineLength], [nLineNumber], [nTabSize], [IWrap]) --> Caracter

Argumento	Obrigat.	Tipo	Descrição
cString	Sim	C	É a string original de onde será extraída a linha.
nLineLength	Não	N	Número de caracteres por linha. Caso não especificado, assume 79.
nLineNumber	Não	N	Número da linha a ser retornada. Caso não especificado, assume 1.
nTabSize	Não	N	Define o número de caracteres para o Tab. Caso não especificado, assume 4.
IWrap	Não	L	Habilita a quebra de linhas de acordo com as palavras. Caso não especificada, fica habilitado.

Retorno	Descrição
Caracter	Caracter. Retorna a linha solicitada, caso não exista o número de linhas, retorna uma string em branco.

Descrição

MemoLine() é uma função que retorna uma linha específica. Caso não exista a linha especificada, retorna um string em branco.

Caso a quebra de linhas esteja habilitada (lWrap igual .T.) e a palavra esteja no meio da quebra de linha, esta será colocada na próxima linha.

MemoLine() é usada em conjunto com MLCOUNT() para extrair todas as linhas de um texto.

Exemplos

```
LOCAL nLineLength := 40, nTabSize := 3, lWrap := .T.  
LOCAL nLines, nCurrentLine  
//  
LOCAL cTexto := MEMOREAD( "Texto.TXT" )  
nLines := MLCOUNT(cTexto, nLineLength, nTabSize, lWrap)  
//  
FOR nCurrentLine := 1 TO nLines  
    conout (MEMOLINE(cTexto, nLineLength, nCurrentLine, nTabSize, lWrap))  
NEXT
```

MemoRead

Lê um arquivo texto e retorna uma string.

Sintaxe

MEMOREAD(cFile) --> Caracter

Argumento	Obrigat.	Tipo	Descrição
cFile	Sim	C	Nome do arquivo texto que deverá ser lido. Caminhos podem ser incluídos.

Retorno	Descrição
variável	Caracter. MEMOREAD() retorna uma string com no máximo 65.535 bytes do arquivo informado.

Descrição

MemoRead() lê um arquivo texto e armazena o conteúdo em uma variável string. O arquivo pode conter no máximo 65.535 bytes.

MemoRead() tentará abrir o arquivo compartilhado e somente para leitura.

Caso o arquivo não possa ser aberto, MemoRead retornará uma string vazia ("").

Exemplos

```
cString = MEMOREAD( "Temp.txt" )  
if empty( cString )
```

```
        conout("Erro lendo arquivo")
    endif
```

MemoWrite

Grava uma string para um arquivo em disco.

Sintaxe

MEMOWRITE(*cArquivo* , *cString*) --> Lógico

Argumento	Obrigat.	Tipo	Descrição
cArquivo	Sim	C	É o nome do arquivo onde será gravado, incluindo a extensão e o caminho.
cString	Sim	C	É a string que será gravada no arquivo.

Retorno	Descrição
.T.	O Arquivo foi gravado com sucesso
.F.	Houve falha na criação e gravação do arquivo, verifique ERROR() para obter detalhes o erro.

Descrição

MEMOWRITE() é uma função que grava o conteúdo de uma string em um arquivo em disco.

Podem ser usados caminhos tanto no local (Remote) como no servidor.

Exemplos

```
LOCAL cString := "Teste de gravação..."
IF MEMOWRITE("teste.txt", cString)
    conout("Erro gravando teste.txt")
ENDIF
```

MLCount

Conta o número de linhas de uma String.

Sintaxe

MLCOUNT(*cString* , [*nLineLength*] , [*nTabSize*] , [*lWrap*]) --> Numérico

Argumento	Obrigat.	Tipo	Descrição
cString	Sim	C	É a string original de onde será extraída a linha.
nLineLength	Não	N	Número de caracteres por linha. Caso não especificado, assume 79.
nTabSize	Não	N	Define o número de caracteres para o Tab. Caso não especificado, assume 4.
lWrap	Não	L	Habilita a quebra de linhas de acordo com as palavras. Se não especificado, fica habilitado.

Retorno	Descrição
>=0	Numérico. Número de linhas que a string possui.

Descrição

MLCOUNT() retorna o número de linhas de uma string baseado nos parâmetros informados.

Exemplos

```

LOCAL nLineLength := 40, nTabSize := 3, lWrap := .T.
LOCAL nLines, nCurrentLine
//
LOCAL cTexto := MEMOREAD( "Texto.TXT" )
nLines := MLCOUNT(cTexto, nLineLength, nTabSize, lWrap)
//
FOR nCurrentLine := 1 TO nLines
    conout(MEMOLINE(CustNotes, nLineLength, nCurrentLine, nTabSize, lWrap))
NEXT

```

Month

Converte o valor da data para o número do mês.

Sintaxe

MONTH(dData) --> Numérico

Argumento	Obrigat.	Tipo	Descrição
dData	Sim	D	É o valor da data a ser convertido

Retorno	Descrição
>=0 e <=12	Para uma data válida.

0	Se a data for nula ou inválida
---	--------------------------------

Descrição

MONTH() é uma função de conversão que extrai da data o valor numérico do mês.

CMONTH() é uma função semelhante que retorna o nome do mês a partir do valor de *dData*.

Exemplos

Estes exemplos retornam o mês da data do sistema:

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

MsCompress

Compacta um ou vários arquivos em um único arquivo com extensão .MZIP.

Sintaxe

MSCOMPRESS(*cArq*, [*cDestino*], [*cSenha*]) --> ISucesso

Ou

MSCOMPRESS(*aArquivos*, [*cDestino*], [*cSenha*]) --> ISucesso

Argumento	Obrigat.	Tipo	Descrição
<i>cArq</i>	Sim	C	Nome do Arquivo a ser compactado.
<i>aArquivos</i>	Sim	A	Nomes dos arquivos a serem compactados.
<i>cDestino</i>	Não	C	Nome do Arquivo destino, caso a extensão seja omitida será assumido .MZIP, se não for informado assumirá o mesmo nome do <i>cArq</i> com extensão .MZIP ou o nome do 1º. Arquivo no Array <i>aArquivos</i> .
<i>cSenha</i>	Não	C	Senha a ser utilizada para criptografar o arquivo.

Retorno	Descrição
.T.	A compactação foi executada com sucesso.
.F.	Erro na compactação, verifique o espaço disponível para compactação.

Descrição

MSCOMPRESS() compacta os arquivos informados em um único arquivo com extensão default .MZIP. O formato é proprietário e multiplataforma.

Caso a senha seja informada apenas com a senha poderemos descompactar os arquivos.

A função para descompactação é a MSDECOMP().

Tanto arquivos no local (Remote) como no Servidor são aceitos.

Exemplos

```
// Exemplo 1 à Compacta apenas um arquivo
lRes := MSCOMPRESS( "AP6SRV.EXE", "AP6SRV.MZIP" )
// Exemplo 2 à Compacta um diretório com senha
aNome := {}
ADIR( "*.DBF", aNome )
lRes := MSCOMPRESS( aNome, "ArqComp.MZIP", "SENHA" )
```

MsCRC32

Calcula um CRC de uma string.

Sintaxe

MSCRC32(*cString*) --> nCRC

Argumento	Obrigat.	Tipo	Descrição
<i>cString</i>	Sim	C	String de onde será calculado um CRC32, é garantido que para a mesma string sempre se obterá um mesmo número, porém, não é garantido que para strings diferentes, os números sejam sempre diferentes.

Retorno	Descrição
nCRC	Um número indicando o CRC da string informada.

Descrição

MSCRC32() calcula um CRC de uma string informada e retorna um número com esse cálculo.

Note que strings iguais retornam CRC iguais, porém, nem sempre strings diferentes retornam CRC diferentes.

Exemplo

```
cString := MEMOREAD( "ARQ.TXT" )  
? MSCRC32( cString )
```

MsCRC32Str

Calcula um CRC de uma string, retornando em formato String.

Sintaxe

MSCRC32STR(*cString*) --> cCRC.

Argumento	Obrigat.	Tipo	Descrição
<i>cString</i>	Sim	C	String de onde será calculado um CRC32, é garantido que para a mesma string sempre obterá um mesmo número, mas não é garantido que para strings diferentes os números sejam sempre diferentes

Retorno	Descrição
nCRC	Uma string com o CRC da string informada

Descrição

MSCRC32STR() calcula um CRC de uma string informada e retornando uma string com esse cálculo.

Note que strings iguais retornam CRC iguais, porém nem sempre strings diferentes retornam CRC diferentes.

Exemplo

```
cString := MEMOREAD( "ARQ.TXT" )  
? MSCRC32STR( cString )
```

MsDecomp

Descompacta arquivos no formato .MZIP (Microsiga Zip).

Sintaxe

MSDECOMP(*cArqZip*, *cPathDestino*, [*cSenha*]) --> ISucesso.

Argumento	Obrigat.	Tipo	Descrição
<i>cArq</i>	Sim	C	Nome do Arquivo a ser descompactado.
<i>cPathDestino</i>	Não	C	Diretório onde os arquivos deverão ser descompactados. Note que podem ser incluídos caminhos do servidor como caminhos locais (Remote).
<i>cSenha</i>	Não	C	Senha a ser utilizada para descriptografar o arquivo.

Retorno	Descrição
.T.	A descompactação foi executada com sucesso.
.F.	Erro na compactação, verifique o espaço disponível para descompactação.

Descrição

MSDECOMP() descompacta o arquivo informado em um diretório. O Formato é proprietário, e multi-plataforma, suporta apenas arquivos compactados pela função MSCOMPRESS().

Caso o arquivo seja protegido por senha, apenas com a senha poderemos descompactá-lo.

A função para compactação é a MSCOMPRESS().

Tanto arquivos no local (Remote) como no Servidor são aceitos.

Exemplo

```
// Exemplo 1 à Descompacta no servidor  
lRes := MSDECOMP( "AP6SRV.MZP", "TEMP" )  
// Exemplo 2 à Descompacta no local ( Remote )  
lRes := MSCOMPRESS( "c:\ArqComp.MZP", "SENHA" )
```

OrdCondSet

Seta a condição e o escopo para a ordem corrente.

Sintaxe

ORDCONDSET([*cForCond*],..., [*bEval*],..., [*IDescendente*],...) --> Lógico

Argumento	Obrigat.	Tipo	Descrição
cForCond	Não	C	Expressão na forma textual a ser resolvida para verificar se o registro em questão está dentro do escopo definido
bEval	Não	Bloco de Código	Expressão na forma executável a ser resolvida para cada registro processado. Este bloco deve retornar tipo lógico
IDescendente	Não	L	Especifica se a ordem deve ser crescente ou decrescente

Retorno	Descrição
.F.	Não conseguiu setar o filtro
.T.	Filtro setado com sucesso

Descrição

Esta função é utilizada para setar uma ordem de registro que esteja dentro de um escopo e de uma condição especificada. Se todos os parâmetros forem omitidos serão aceitos todos os registros da ordem.

Através do primeiro parâmetro (*cForCond*) é possível especificar o escopo ao qual o registro deve pertencer para estar dentro do filtro. Através do parâmetro *bEval*, pode-se definir um bloco de código que deve retornar .T. para que o registro pertença ao filtro a ser setado.

Se o parâmetro *IDescendente* for omitido, a ordem estará crescente, mas se tiver valor .T. será decrescente.

Exemplo

```
// Este exemplo mostra como se pode usar o ORDCONDSET para executar um filtro com idade
entre 20 e 30 anos e nome Joao:
USE Cliente VIA "DBFCDX" NEW
ORDCONDSET("Idade>20 .AND. Idade<30",,,,{|Nome = "Joao"})
```

OrdCreate

Cria uma ordem em determinado arquivo de índice.

Sintaxe

ORDCREATE([*cIndice*],[*cOrdem*], *cExpChave*, [*bExpChave*], [*IUnico*]) --> Nil

Argumento	Obrigat.	Tipo	Descrição
<i>cIndice</i>	Não *	C	Nome do arquivo de índice a ser criada a ordem.
<i>cOrdem</i>	Não *	C	Nome da ordem a ser criada.
<i>cExpChave</i>	Sim	C	Expressão das chaves da ordem a ser criada na forma de string.
<i>bExpChave</i>	Não	Bloco de Código	Expressão das chaves da ordem a ser criada na forma executável.
<i>IUnico</i>	Não	L	Cria índice como único (o padrão é .F.).

*Obs: Os dois primeiros parâmetros são opcionais, mas pelo menos um deles tem que estar especificado.

Retorno	Descrição
NIL	Sem retorno

Descrição

Esta função é utilizada para criar uma nova ordem em determinado arquivo de índice com o nome especificado através do primeiro parâmetro, sendo que, se o mesmo existir, é apenas acrescentada a ordem, mas, caso contrário, é criado o arquivo.

Para tanto, são executados os passos a seguir:

- 1- Salva fisicamente as alterações ocorridas na tabela corrente;
- 2- Fecha todos os arquivos de índice abertos;
- 3- Cria o novo arquivo de índice se não existir;
- 4- Cria a nova ordem;
- 5- Seta a nova ordem como a ordem corrente;
- 6- 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 da ordem, pois na criação de índices no Ctree é alterada a estrutura da tabela, sendo necessário que a tabela esteja aberta em modo exclusivo.

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
ORDCREATE ( "\teste\ind2.cdx" , "Nome+End", { || Nome+End }, .T. )
// Este exemplo mostra como se pode criar nova ordem (Tag2) sobre o campo End que
// aceitará duplicação e no arquivo de índice já existente:
USE Cliente VIA "DBFCDX" NEW
ORDCREATE ( "\teste\ind2.cdx" , "Tag2", "End", { || End } )
```

OrdDescend

Verifica ou altera a condição (crescente/decrescente) de uma ordem.

Sintaxe

ORDDSCEND([*cOrdem* | *nPosicao*], [*cArqIndice*], [*IDecrescente*]) --> Lógico

Argumento	Obrigat.	Tipo	Descrição
<i>cOrdem</i>	Não	C	Nome da ordem a ser alterada
<i>nPosicao</i>	Não	N	Posição da ordem na lista a ser alterada
<i>cArqIndice</i>	Não	C	Nome do arquivo de índice
<i>IDecrescente</i>	Não	L	Determina se a ordem será decrescente (.T.) ou crescente (.F.)

Retorno	Descrição
.F.	A ordem do parâmetro está na forma crescente
.T.	A ordem do parâmetro está na forma decrescente

Descrição

Esta função pode ser utilizada para apenas verificar o estado da ordem atual, se não for especificado o parâmetro *IDecrescente*.

Quando é especificado o parâmetro *!Descend*, o estado da ordem é modificado, sendo que a função retorna ao estado anterior da ordem em questão.

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.

Exemplo

```
// Este exemplo mostra como o ORDBAGNAME pode encontrar o nome de diferentes índices
// através da posição da ordem na lista:
USE Cliente VIA "DBFCDX" NEW
ORDLISTADD ("teste\ind1.cdx", "NOME") // A ordem é criada na forma crescente
ORDDDESCEND() // Retorna: .F.
ORDDDESCEND(, .T.) // Retorna: .F., mas seta a ordem atual (NOME) como decrescente
ORDDDESCEND() // Retorna: .T.
ORDLISTADD ("teste\ind2.cdx", "NOME")
ORDDDESCEND("NOME", "ind1", .F.) // Retorna: .T.
ORDDDESCEND("NOME", "ind1",) // Retorna: .F.
```

OrdKey

Verifica qual a expressão de chave da ordem.

Sintaxe

ORDKEY([*cOrdem* | *nPosicao*], [*cArqIndice*]) --> Caracter

Argumento	Obrigat.	Tipo	Descrição
<i>cOrdem</i>	Não	C	Nome da ordem a ser alterada
<i>nPosicao</i>	Não	N	Posição da ordem na lista
<i>cArqIndice</i>	Não	C	Nome do índice

Retorno	Descrição
""	Não existe ordem corrente.
<i>cExpOrdem</i>	Expressão de chave da ordem ativa ou especificada pelos parâmetros.

Descrição

Esta função é utilizada para verificar 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 carácter.

Exemplo

```
// Este exemplo mostra como o ORDKEY() pode recuperar a expressão da ordem atual:  
USE Cliente NEW  
INDEX ON Nome+Cod TO Ind1 FOR Nome+Cod > "AZZZZZZZ"  
ORDKEY("Ind1") // Retorna: Nome+Cod
```

OrdListAdd

Acrescenta uma ou mais ordens à lista.

Sintaxe

ORDLISTADD(*cArqIndice*, [*cOrdem*]) --> Nil

Argumento	Obrigat.	Tipo	Descrição
<i>cArqIndice</i>	Sim	C	Nome do índice a ser acrescentado, com ou sem diretório e extensão
<i>cOrdem</i>	Não	C	Nome da ordem a ser acrescentada

Retorno	Descrição
NIL	Sem retorno

Descrição

Esta função é utilizada para acrescentar uma ou mais ordens de determinado índice na lista de ordens ativas da área de trabalho.

Quando são especificados os dois argumentos (índice e ordem), é acrescentada apenas a ordem especificada nos parâmetros à lista e a mesma torna-se ativa.

Quando é especificado apenas o primeiro parâmetro, são acrescentadas todas as ordens contidas no arquivo de índice especificado neste parâmetro à lista, e a primeira ordem torna-se ativa.

Para se utilizar arquivos de extensão padrão do RDD, este dado pode ser omitido no primeiro parâmetro, mas, caso contrário, deve ser especificado.

Exemplo

```
// Este exemplo mostra como se pode acrescentar uma ordem específica ou todas as ordens
de determinado arquivo de índice à lista:
USE Cliente VIA "DBFCDX" NEW
OrdListAdd ("teste\ind1.cdx", "NOME") // Acrescenta apenas a ordem NOME
OrdListAdd ("teste\ind2.cdx") // Acrescenta todas as ordens do arquivo ind2.cdx
```

PRow

Informa ou muda a linha corrente de impressão.

Sintaxe

PRow([*nNewLine*]) --> Numérico

Argumento	Obrigat.	Tipo	Descrição
nNewLine	Não	N	Informar a nova linha de impressão.

Retorno	Descrição
N	Número da linha corrente de impressão

Descrição

PRow() pode ser utilizado para informar a linha corrente de impressão ou para modificá-la. Se a nova linha informada for menor que a corrente, isto provocará um salto de página na impressão.

Exemplo

```
@ PRow( ), 10 PSAY "Minha Linha nesta pagina e " + str( PRow( ) )
PRow( PRow( )+2 ) // Pula 2 linhas
@ PRow( ), 10 PSAY "Minha Linha nesta pagina e " + str( PRow( ) )
PRow( 0 ) // Salta a pagina
```

Pack

Remove todos os registros deletados da tabela.

Sintaxe

PACK

Descrição

Este comando apaga (fisicamente) todos os registros deletados da tabela corrente.

Exemplo

```
// Este exemplo demonstra como se pode utilizar a função DBDELETE\(\) para marcar alguns registros como deletados e o comando PACK para deletá-los fisicamente.  
USE Clientes NEW  
DBGOTO(100)  
DBDELETE()  
DBGOTO(105)  
DBDELETE()  
DBGOTO(110)  
DBDELETE()  
PACK
```

PCol

Informa ou muda a coluna corrente de impressão.

Sintaxe

PCOL([*nNewCol*]) --> Numérico

Argumento	Obrigat.	Tipo	Descrição
<i>nNewCol</i>	Não	N	Informar a nova coluna de impressão.

Retorno	Descrição
N	Número da coluna corrente de impressão.

Descrição

PCOL() pode ser utilizado para informar a coluna corrente de impressão ou para modificá-la. Se for informada uma coluna menor que a corrente, serão impressos caracteres de BACKSPACE chr(8) para forçar o retorno do carro de impressão em impressoras matriciais.

Exemplo

```
PCOL( 10 )
@ ROW( ), PCOL( ) PSAY "Minha Linha nesta pagina e " + str( PROW( ) )
PROW( PROW( )+2 ) // Pula 2 linhas
PCOL( 10 )
@ PROW( ), PCOL( ) PSAY "Minha Linha nesta pagina e " + str( PROW( ) )
PROW( 0 ) // Salta a pagina
```

PreparePrint

Prepara o relatório para o início da impressão.

Sintaxe

PREPAREPRINT(*IWindows*, *cPrinterName*, *IDisco*, *cFile*, *I_Lpd*, *nLeftMargin*) --> NIL

Argumento	Obrigat.	Tipo	Descrição
IWindows	Sim	L	Define se a impressão será via spool do Windows ou diretamente em porta de impressão.
cPrinterName	Sim	C	Informa o nome da impressora que será acionada. Se <i>IWindows=.T.</i> , deve-se informar o nome da impressora Windows. Se <i>IWindow=.F.</i> , deve-se informar o nome da porta de impressão "LPT1", "LPT2".
IDisco	Sim	L	Informa se a impressão será direcionada para gravação em arquivo. Se <i>IDisco=.T.</i> , <i>IWindows</i> e <i>cPrinterName</i> , não terão efeito então deve-se informar o nome do arquivo de saída em <i>cFile</i> .
cFile	Sim	C	Informa o nome de arquivo que será gerado se <i>IDisco=.T.</i> .
I_Lpd	Não	L	Aplicável somente para Server executando em ambiente Unix. Informa que a impressão será direcionada para disco e logo em seguida direcionada para o LPD (daemon de impressão do Unix).
nLeftMargin	Não	N	Margem esquerda do relatório para ajuste para encadernação em cm.

Retorno	Descrição
NIL	Sem retorno.

Descrição

PREPAREPRINT() ajusta a impressão do relatório propriamente dita, define se utilizará o spool do Windows ou se o relatório será direcionado para arquivo. Pode-se ajustar a margem esquerda do relatório para encadernações.

Quando estamos utilizando um servidor em ambiente Unix, pode-se configurar a saída do relatório para utilizar o LPD (Line Printer Daemon) do Unix. Ative o parâmetro `_lpd` para `.T.`, e no arquivo de configuração do Server (AP6SRV.INI), crie a entrada abaixo:

```
[SERVERPRINTERS]
```

```
PRINTERSNAME=lp1,lp2[...lp3]
```

Onde PRINTERSNAME é uma lista com os nomes das impressoras cadastradas no `\etc\printcap`. Para mais informações veja manual do Unix para configuração de impressoras. Lembrando que o Unix é case-sensitive quando trata nomes e arquivos.

Exemplo

```
// Envia a impressao para a impressora "LaserJet4.." e configura a margem esquerda para 1
cm para direita
PreparePrint( .T., "LaserJet4 in //advpr1",.F.,",",.F.,1 )
```

PrnFlush

Força envio do buffer de impressão para a impressora.

Sintaxe

```
PRNFLUSH( ) --> NIL
```

Retorno	Descrição
NIL	Sem retorno

Descrição

Nas impressões, o Protheus bufferiza a impressão em páginas para enviar todo o conteúdo para a impressora de forma otimizada. PRNFLUSH() força o envio do buffer de dados para a impressora antes do previsto pela otimização de impressão.

Exemplo

```
SETPRC(0,0) // inicia impressão na linha 0 coluna 0
PCOL( 10 ) //muda para coluna 10
@ ROW( ), PCOL( ) PSAY "Minha Linha nesta pagina e " + str( PROW( ) )
PRNFLUSH( ) // Força impressão antes do termino da pagina.
PROW( PROW( )+2 ) // Pula 2 linhas
PCOL( 10 )
@ PROW( ), PCOL( ) PSAY "Minha Linha nesta pagina e " + str( PROW( ) )
PROW( 0 ) // Salta a pagina
FECHAREL( ) // Fecha impressão
```

RDDSetDefault

Modifica ou verifica o RDD padrão.

Sintaxe

RDDSetDefault ([*cNovoRddPadrão*]) --> Caracter

Argumento	Obrigat.	Tipo	Descrição
<i>cNovoRddPadrão</i>	Não	C	Novo nome do RDD a ser definido como padrão.

Retorno	Descrição
<i>cAntigoRddPadrão</i>	Nome do RDD padrão corrente

Descrição

Esta função 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.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o RDDSETDEFAULT para alterar o valor do RDD padrão.  
RDDSETDEFAULT("CTREECDX") // Retorna: DBFCDX  
RDDSETDEFAULT() // Retorna: CTREECDX
```

RealRDD

Retorna qual é o driver que realmente está sendo utilizado para abertura dos arquivos locais.

Sintaxe

REALRDD() --> Caracter

Retorno	Descrição
---------	-----------

"ADS"	Advantage Local Server
"ADSSERVER"	Advantage Database Server
"CTREE"	Ctree - Faircom
"CODEBASE"	Codebase

Descrição

O driver "DBFCDX" pode ser utilizado para a abertura de arquivos Codebase, Advantage Database Server, Advantage Local Server ou Ctree. Depende do que está especificado na chave LOCALFILES do ambiente utilizado, no arquivo de inicialização do servidor.

REALRDD() retorna o nome do driver que realmente está sendo utilizado para abertura dos arquivos locais.

Exemplo

```
if Real Rdd()="CODEBASE"
conout("Mudar abertura dos arquivos para ADS.")
endif
```

Recall

Altera o estado deletado de alguns registros.

Sintaxe

RECALL [FOR *CondFor*] [WHILE *CondWhile*] [NEXT *nRecs*] [RECORD *nRecno*] [REST] [ALL]

Argumento	Obrigat.	Tipo	Descrição
CondFor	Sim		Expressão em ADVPL a ser resolvida para o registro ser aceito.
CondWhile	Não		Expressão em ADVPL que determina quando a busca deve parar (quando a expressão retornar .F..
nRecs	Não	N	Quando registros devem ser verificados.
nRecno	Não	N	Número do recno do registro a ser verificado.

Retorno	Descrição
NIL	Sem retorno

Descrição

Este comando é utilizado para retirar a marca de registro deletado de alguns registros.

Para ser executado, os registros em questão devem estar bloqueados ou a tabela deve estar aberta em modo exclusivo. Se o registro não estiver deletado, este comando não faz nada.

Ele é o oposto da função [DBDELETE](#), que marca o registro atual como deletado.

A seleção dos registros a serem alterados é feita da seguinte forma:

- se todas as opções forem omitidas ele é executado apenas para o registro atual;
- se for especificada uma condição de "FOR", o escopo abrange todos os arquivos ("ALL");
- o escopo vai até quando a condição de "WHILE" retorna .F.;
- o parâmetro "NEXT *nRecs*" determina quantos registros devem ser alterados;
- o parâmetro "RECORD *nRecno*" especifica qual recno do registro deve ser alterado;
- a opção "REST" determina que serão processados os registros a partir do registro atual. Caso seja omitida, o comando começa a processar a partir do primeiro registro.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o RECALL para retornar o estado do
registro atual para normal.
USE Cliente
DBGOTO(100)
DBDELETE()
DELETED() // Retorna: .T.
RECALL
DELETED() // Retorna: .F.
// Este exemplo demonstra como se pode utilizar o RECALL para retornar o estado de todos
os registros que apresentam idade>30.
USE Cliente
RECALL FOR Idade>30
// Este exemplo demonstra como se pode utilizar o RECALL para retornar o estado de todos
os registros com idade maior que 30 e Nome = Joao a partir do registro atual.
USE Cliente
RECALL FOR Idade>30 WHILE Nome="Joao" REST
```

RecSize

Verifica o tamanho do registro da tabela corrente.

Sintaxe

RECSIZE() --> Numérico

Retorno	Descrição
nBytes	<i>Tamanho do registro da tabela corrente em número de bytes.</i>
0	Não há tabela corrente.

Descrição

Esta função calcula o tamanho do registro da tabela corrente somando os tamanhos de cada campo mais um byte da flag de registro deletado mais quatro bytes do campo recno.

Ela pode ser utilizada em conjunto com as funções [Header](#) e RecCount para calcular o tamanho ocupado no disco pela tabela corrente, pois o tamanho será Header+RecSize*RecCount.

Exemplo

```
// Este exemplo calcula o tamanho ocupado pela tabela "AA1990.DBF" no disco (número de bytes):
USE "\DADOSADV\AA1990.DBF" SHARED NEW
nCabecalho := HEADER()
nDados := RECSIZE() * RECCOUNT()
nTamanhoTotal := nCabecalho + nDados
```

Reindex

Reconstrói todos os índices abertos da área de trabalho corrente.

Sintaxe

REINDEX

Descrição

Este comando reconstrói todos os índices da área de trabalho corrente e posiciona as tabelas no primeiro registro lógico.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o DBREINDEX para reconstruir os índices depois que um novo índice foi gerado.
USE Clientes NEW
DBSETINDEX("IndNome")
REINDEX
```

Replace

Modifica o valor de alguns campos e registros da tabela corrente.

Sintaxe

REPLACE *Campo* WITH *Exp* [, *Campo2* WITH *Exp2* ...] [FOR *CondFor*] [WHILE *CondWhile*] [NEXT *nRecs*] [RECORD *nRecno*] [REST] [ALL]

Argumento	Obrigat.	Tipo	Descrição
<i>Campo</i>	Sim		Nome do campo a ser alterado
<i>Exp</i>	Sim		Expressão em ADVPL com valor a ser colocado no campo especificado
<i>CondFor</i>	Não		Expressão em ADVPL a ser resolvida para que o registro seja alterado
<i>CondWhile</i>	Não		Expressão em ADVPL que determina quando a alteração deve parar (quando a expressão retornar .F.)
<i>nRecs</i>	Não	N	Quando registros devem ser alterados
<i>nRecno</i>	Não	N	Número do recno do registro a ser alterado

Descrição

Este comando é utilizado para alterar o valor de determinado campo em alguns registros da tabela corrente, onde o campo escolhido recebe o valor da expressão para os registros dentro do escopo definido.

Se não forem especificadas as condições para que o registro seja alterado, modifica toda a tabela, como se estivesse especificada a opção "ALL".

Pode-se especificar um escopo para que os registros sejam alterados através das opções "FOR *CondFor*" e "WHILE *CondWhile*".

Pode-se também limitar o número de registros a serem alterados através da opção "NEXT *nRecs*" e determinar que a alteração dos registros deve-se iniciar a partir do registro atual com "REST", mas, caso contrário, o comando executa um [DBGOTOP\(\)](#) antes de iniciar a alteração.

Se é desejado alterar apenas determinado registro pode-se defini-lo através da especificação do recno com "RECORD *nRecno*".

Exemplo

```
// Este exemplo demonstra como utilizar o comando REPLACE alterar todos os registros dentro do escopo, onde o campo "Valor" receberá "Valor1+Valor2-Valor3*0.1" e o campo "DiaTran" receberá a data atual. Este escopo é definido por Idade > 20, até que o nome seja maior ou igual a "VVV", começa a deleção a partir do registro atual e marca apenas 10 registros:
```

```
USE Cliente VIA "CTREECDX" NEW
REPLACE Valor WITH Valor1+Valor2-Valor3*0.1, DiaTran WITH Date() FOR Idade>20 WHILE
Nome<"VVV" NEXT 10 REST
```

RLock

Bloqueia o registro corrente da tabela ativa.

Sintaxe

RLOCK() --> Lógico

Retorno	Descrição
.F.	Não conseguiu bloquear o registro. Principal motivo: o registro já está bloqueado por outro usuário.
.T.	O registro foi bloqueado com sucesso.

Descrição

Esta função 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.

Exemplo

//Este exemplo utiliza a função RLOCK() para deletar o registro com o nome "Joao" da tabela de Clientes indexada por Nome:

```
USE Clientes INDEX Nome SHARED NEW
SEEK "Joao"
IF FOUND()
    IF RLOCK()
        DELETE
        MessageBox("Joao deletado","OK", 0)
    ELSE
        MessageBox("Registro utilizado por outro usuário","Erro", 0)
    ENDIF
ELSE
    MessageBox("Registro não encontrado","Erro", 0)
ENDIF
CLOSE
//Este exemplo mostra como se pode bloquear um registro sem que ele esteja na tabela
corrente
USE VENDAS NEW
USE CLIENTES NEW
//
IF !VENDAS->(RLOCK())
    MessageBox("Registro utilizado por outro usuário","Erro", 0)
    BREAK
ENDIF
```

Seconds

Retorna o número de segundos decorridos desde a meia-noite.

Sintaxe

SECONDS() --> Numérico

Retorno	Descrição
>=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.

Descrição

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.

Exemplos

Este exemplo compara TIME() e SECONDS():

```
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()
.
. <statements>
.
nElapsed:= SECONDS() - nStart
cElapsed := LTRIM(STR(nElapsed)) + " seconds"
```

Seek

Encontra um registro com determinado valor da chave do índice.

Sintaxe

SEEK Exp [SOFTSEEK]

Argumento	Obrigat.	Tipo	Descrição
Exp	Sim		Expressão em ADVPL a ser resolvida para o registro ser encontrado

Retorno	Descrição
NIL	Sem retorno

Descrição

Este comando é utilizado para encontrar um registro com determinado valor da expressão de chave de índice.

Antes da chamada do SEEK deve-se certificar de que existe uma ordem ativa no momento com os campos que se deseja pesquisar o valor.

Caso a expressão possua apenas um campo numérico, o primeiro parâmetro deve ser do tipo numérico, mas nos demais casos deve-se utilizar um valor do tipo caracter para este parâmetro (mesmo se forem apenas dois campos numéricos ou do tipo data).

Quando for especificada a opção "SOFTSEEK", mesmo que a expressão pesquisada não encontrar nenhum registro com este valor, a tabela será posicionada no próximo valor maior que o especificado no primeiro parâmetro, mas mesmo posicionando no próximo valor, esta função retornará .F. (pois não encontrou).

Quando não for especificado este valor ou estiver .F. e falhar o valor de pesquisa, a tabela será posicionada em LASTREC + 1 e será setada a flag de EOF.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o SEEK para busca de valores numéricos.
USE Clientes NEW
ORDLISTADD ("/teste/ind1.cdx") // Expressão é Num (campo numérico)
SEEK 100 // Retorna: .F.
EOF() // Retorna: .T.
SEEK 100 SOFTSEEK // Retorna: .F.
EOF() // Retorna: .F. (pois o softseek posicionou no próximo registro)
// Este exemplo demonstra como se pode utilizar o SEEK para percorrer todos os registros
de Clientes com o nome joao e vencimentos a partir de janeiro de 2001.
USE Clientes NEW
ORDLISTADD ("/teste/ind2.cdx") // Expressão é Nome+Venc (campo caracter + data)
SEEK "      joao200101" SOFTSEEK // Procura a primeira ocorrência de Nome "joao" e
vencimento maior que Janeiro de 2001
WHILE !EOF() .AND. Nome == "      joao"
    DBSKIP()
ENDDO
```

Select

Seleciona nova área de trabalho.

Sintaxe

SELECT *nArea* | *Alias*

Argumento	Obrigat.	Tipo	Descrição
<i>nArea</i>	Sim	N	Número da área de trabalho a ser selecionada.
Alias	Sim		Nome da identificação da área de trabalho a ser selecionada (Alias).

Descrição

Este comando é utilizado para selecionar uma nova área de trabalho para deixá-la ativa.

Exemplo

```
// Este exemplo demonstra como se pode utilizar o SELECT alterar a área corrente.
USE Clientes ALIAS a1
USE Clientes2 ALIAS a2
SELECT a1 // ou SELECT 1
```

Set Filter

Seta ou cancela uma condição de filtro.

Sintaxe

SET FILTER TO [*Condição*]

Argumento	Obrigat.	Tipo	Descrição
<i>Condição</i>	Não		Expressão em ADVPL a ser setada como filtro na ordem corrente

Retorno	Descrição
NIL	Sem retorno

Descrição

Este comando é utilizado para setar um filtro nos registros da tabela corrente especificado através da condição especificada.

Quando um registro não está dentro do filtro setado ele continua existindo fisicamente, mas não logicamente (nas funções de manipulação de banco de dados como [DBGOTOP](#), [DBSEEK](#), [DBSKIP](#), etc).

Se o comando é chamado com condição em vazio, será cancelado o filtro existente.

Exemplo

```
// Este exemplo demonstra como utilizar o comando SET FILTER para setar novas expressões de filtro e retirá-las:  
USE Cliente VIA "CTREECDX" NEW  
SET ORDER TO 2 // Seta a ordem de nome Nome do índice Ind1  
SET FILTER TO Idade>30 // Filtra os registros com Idade menor que 30  
SET FILTER TO
```

Set Index

Acrescenta todas as ordens de um ou mais arquivos de índice à lista.

Sintaxe

SET INDEX TO [*ArqIndices*] [*ADDITIVE*]

Argumento	Obrigat.	Tipo	Descrição
<i>ArqIndices</i>	Não		Nome dos arquivos de índice a serem acrescentados à lista de ordens.

Retorno	Descrição
NIL	Sem retorno

Descrição

Este comando é utilizado para acrescentar uma ou mais ordens de determinado índice na lista de ordens ativas da área de trabalho quando se especifica "ADDITIVE".

Quando o mesmo é omitido, a lista de ordens é esvaziada para depois ser acrescentada às ordens dos índices.

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.

Quando mais de um arquivo de índice é selecionado, a ordem que torna-se ativa é a primeira ordem do primeiro arquivo.

Quando o comando é utilizado sem nenhum parâmetro, todas as ordens da lista são apagadas.

Exemplo

```
// Este exemplo demonstra como utilizar o comando SET INDEX para acrescentar novas ordens a lista e retirá-las depois:  
USE Cliente VIA "CTREECDX" NEW  
SET INDEX TO ind1 // Inicializa a lista com as ordens do arquivo de índice "ind1"  
SET INDEX TO ind2 ADDITIVE // Acrescenta as ordens do arquivo de índice "ind2" na lista  
SET INDEX TO ind3 // Limpa a lista e inicializa com as ordens do arquivo "ind3"  
SET INDEX TO // Limpa a lista de ordens
```

Set Order

Seleciona uma ordem ativa da área de trabalho.

Sintaxe

SET ORDER TO [*nPosição* | [TAG *cOrdem*] [IN *cÍndice*]]

Argumento	Obrigat.	Tipo	Descrição
<i>nPosição</i>	Não	N	Posição da ordem na lista de ordens ativas
<i>cOrdem</i>	Não		Nome da ordem a ser setada
<i>cÍndice</i>	Não		Nome do arquivo de índice a ser ao qual pertence a ordem a ser setada

Retorno	Descrição
NIL	Sem retorno

Descrição

Este comando pode ser utilizado apenas para retornar a tabela corrente à ordem original (recno), se não for especificado nenhum parâmetro.

Também pode ser utilizado para selecionar uma nova ordem para a tabela corrente através da posição da ordem na lista de ordens ativas com o primeiro parâmetro ou através do nome da ordem especificado "TAG *cOrdem*".

Para evitar nomes de ordens duplicados, pode-se especificar a qual arquivo de índice pertence com "IN *cÍndice*".

Exemplo

```
// Este exemplo demonstra como utilizar o comando SET ORDER para setar novas ordens e
retirá-las:
USE Cliente VIA "CTREECDX" NEW
SET ORDER TO TAG Nome IN Ind1 // Seta a ordem de nome Nome do índice Ind1
SET ORDER TO 3 // Seta a terceira ordem da lista
SET ORDER TO // Retira as ordens, setando a ordem natural da tabela
```

SetPrc

Configura a linha e coluna correntes de impressão.

Sintaxe

SETPRC(*nLin*, *nCol*) --> NIL

Argumento	Obrigat.	Tipo	Descrição
nLin	Sim	N	Informar a nova linha de impressão
nCol	Sim	N	Informar a nova coluna de impressão

Retorno	Descrição
NIL	Sem retorno.

Descrição

SETPRC() modifica a linha e coluna atuais de impressão assim como PCOL() e PROW() chamados separadamente.

Exemplo

```
SETPRC(0,0) // inicia impressão na linha 0 coluna 0
PCOL( 10 ) //muda para coluna 10
@ ROW( ), PCOL( ) PSAY "Minha Linha nesta pagina e " + str( PROW( ) )
PROW( PROW( )+2 ) // Pula 2 linhas
PCOL( 10 )
@ PROW( ), PCOL( ) PSAY "Minha Linha nesta pagina e " + str( PROW( ) )
PROW( 0 ) // Salta a pagina
```

Skip

Desloca a tabela para outro registro.

Sintaxe

SKIP [*nRegistros*] [*nArea* | ALIAS *Alias*]

Argumento	Obrigat.	Tipo	Descrição
<i>nRegistros</i>	Não	N	Número de registros a ser deslocados
<i>nArea</i>	Não	N	Número da área de trabalho
<i>Alias</i>	Não		Nome da identificação da área de trabalho

Retorno	Descrição
NIL	Sem retorno

Descrição

Este comando é utilizado para deslocar para outro registro a partir do registro atual.

O parâmetro especifica quantos registros lógicos devem ser deslocados a partir do corrente. Se for positivo desloca em direção ao final da tabela, se for negativo ao início da tabela e, caso seja omitido, irá para o próximo registro (o padrão é 1). Caso passe do início da tabela, posiciona no primeiro registro e seta BOF, caso passe do final da tabela, posiciona no registro LASTREC + 1 e seta EOF.

O padrão é ser utilizado para a tabela corrente, mas pode ser especificada outra área de trabalho para executá-lo através do número "*nArea*" ou nome "ALIAS *Alias*".

Exemplo

```
// Este exemplo mostra como o SKIP pode passar do final da tabela e do início da tabela
DBUSEAREA( .T., "dbfcdxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F. )
DBGOBOTTOM()
EOF() // retorna .F.
SKIP
EOF() // retorna .T.
DBGOTOP()
BOF() // retorna .F.
SKIP -1
BOF() // retorna .T.
// Este exemplo mostra como o SKIP pode deslocar 10 registro em relação ao registro
corrente
DBUSEAREA(.T., "dbfcdxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F. )
DBGOTO(100)
SKIP 10
RECNO() // retorna 110
SKIP -10
RECNO() // retorna 100
// Este exemplo mostra como o SKIP pode ser executado em outra area de trabalho.
DBUSEAREA(.T., "dbfcdxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F. )
```

```

DBGOTO(100)
DBUSEAREA(.T., "dbfcdxads", "\dadosadv609\sa1110.dbf", "XXX", .T., .F. )
SKIP 10 ALIAS "SSS"
SSS->RECNO() // retorna 110
SKIP -10 ALIAS "SSS"
SSS->RECNO() // retorna 100

```

SplitPath

Quebra em diversas partes um arquivo.

Sintaxe

SPLITPATH(*cArq*, @[*cDrive*], @[*cCaminho*], @[*cNome*], @[*cExt*]) --> NIL

Argumento	Obrigat.	Tipo	Descrição
<i>cArq</i>	Sim	C	Nome do Arquivo a ser quebrado. Opcionalmente, pode incluir caminho e drive.
<i>cDrive</i>	Não	C	Nome do Drive. Exemplo (C:). Caso o Arquivo informado não possua drive ou o caminho refira-se ao servidor, retorno será uma string em branco.
<i>cCaminho</i>	Não	C	Nome do Caminho. Caso o Arquivo informado não possua caminho, será uma string em branco.
<i>cNome</i>	Não	C	Nome do Arquivo sem a extensão, caso em <i>cArq</i> não seja especificado um nome do Arquivo retornará uma string em branco.
<i>cExt</i>	Não	C	Nome do Arquivo sem a extensão, caso em <i>cArq</i> não seja especificado um Arquivo com extensão retornará uma string em branco

Retorno	Descrição
NIL	Sem retorno

Descrição

SplitPath() divide um caminho completo em todas as suas subpartes; *cArq* não necessita conter todas as partes.

Tanto arquivos locais (Remote) quanto arquivos no servidor, podem ser informados.

O caminho, caso informado, incluirá uma barra como último caracter.

A extensão inclui sempre o ponto (.) antes da extensão.

Todos os parâmetros quando passados devem ser por referência.

Exemplo

```
Local cArq := "C:\TEMP\TESTE.EXE"
Local cDrive, cDir, cNome, cExt
SplitPath( cArq, @cDrive, @cDir, @cNome, @cExt )
? cDrive // Resultado : C:
? cDir // Resultado : \TEMP\
? cNome // Resultado : TESTE
? cExt // Resultado: .EXE
```

TCConType

Define o tipo de conexão que será utilizada entre o Protheus e o TopConnect.

Sintaxe

TCCONTYPE (*cTipo*) --> NIL

Argumento	Obrigat.	Tipo	Descrição
<i>cTipo</i>	Sim	C	Tipo da conexão. Pode ser: "TCPIP" ou "NPIPE"

Retorno	Descrição
NIL	Sem retorno

Descrição

Determina o tipo de conexão que será utilizada entre o Protheus e o TopConnect. O valor é guardado e utilizado nas chamadas seguintes de TCLink.

Exemplo

```
TCConType("NPIPE")
_nCon := TCLink("MSSQL7/TOPSQL", "TOPSERVER1")
If (_nCon < 0)
CONOUT("Falha Conexao TOPCONN - Erro: "+ str(nCon, 10, 0))
EndIf
```

TCDeIFile

Apaga um arquivo de um banco de dados.

Sintaxe

TCDELFILE (*cTabela*) --> Lógico

Argumento	Obrigat.	Tipo	Descrição
<i>cTabela</i>	Sim	C	Nome da tabela que deve ser apagada.

Retorno	Descrição
.T.	Tabela foi excluída com sucesso
.F.	Erro. Geralmente é porque a tabela está sendo utilizada por outro processo.

Descrição

Apaga um arquivo de um banco de dados relacional via TopConnect.

Exemplo

```
If TcCanOpen( "TRB"+SM0->M0_CODIGO+"0" )
TcDeIFile( "TRB"+SM0->M0_CODIGO+"0" )
Endif
```

TCGenQry

Define a execução de uma Query.

Sintaxe

TCGENQRY ([*xPar1*, *xPar2*,], *cQuery*) --> caracter

Argumento	Obrigat.	Tipo	Descrição
<i>xPar1</i> , <i>xPar2</i>	Não	Qualquer	Parâmetros apenas para compatibilização. Não tem função.
<i>cQuery</i>	Sim	C	Contém a expressão da query que se deseja executar.

Retorno	Descrição
---------	-----------

""	Sempre retorna uma string vazia.
----	----------------------------------

Descrição

Esta função determina que a próxima chamada à DBUseArea será a abertura de uma Query e não de tabela.

Exemplo

```
cQuery := 'SELECT X2_CHAVE CHAVE, R_E_C_N_O_ RECNO from SX2990'
dbUseArea(.T., 'TOPCONN', TCGenQry(,cQuery), 'TRB', .F., .T.)
while !Eof()
    // Processa
    conout(TRB->CHAVE)
    dbSkip()
enddo
dbCloseArea()
```

TCISVLock

Verifica se o servidor possui sistema de locks virtuais.

Sintaxe

TCISVLOCK () --> Lógico

Retorno	Descrição
.T.	O servidor possui sistemas de locks virtuais.
.F.	O servidor não possui locks virtuais.

Descrição

Através de locks virtuais, é possível bloquear uma string. A função TCISVLOCK verifica se o servidor TopConnect possui tratamento para locks virtuais.

Exemplo

```
#ifdef TOP
If TCISVLock()
TCVUnLock()
EndIf
#endif
```

TCRefresh

Faz refresh em uma tabela.

Sintaxe

TCREFRESH (*cTabela*) --> NIL

Argumento	Obrigat.	Tipo	Descrição
<i>cTabela</i>	Sim	Lógico	Indica nome da tabela que deve ser feito refresh.

Retorno	Descrição
NIL	Não existe retorno.

Descrição

Faz o refresh de uma tabela, através de uma leitura forçada da tabela no banco de dados. É útil após alterações diretas no banco (delete, insert).

Exemplo

```
cTabela:= "SA1990"
cComando := "Delete " + cTabela + " Where R_E_C_N_O_ > 50000 "
TCSqlExec(cComando)
TCRefresh(cTabela)
```

TCSetBuff

Esta função foi mantida apenas para compatilização, não sendo utilizada no AP6.

TCSetConn

Altera a conexão corrente.

Sintaxe

TCSETCONN(*nConexaoCorrente*) --> Lógico

Argumento	Obrigat.	Tipo	Descrição
<i>nConexaoCorrente</i>	Sim	Numérico	Indica o número da conexão que deve se tornar a corrente. Este número foi retornada pela função TCLINK.

Retorno	Descrição
---------	-----------

.T.	Conexão corrente trocada com sucesso
.F.	Conexão não encontrada

Descrição

Altera a conexão corrente. Novas tabelas abertas ou criadas utilizarão esta conexão para realizar a operação. É útil quando se tem mais de uma conexão com o TopConnect.

Exemplo

```
_nCon1 := TCLink("MSSQL7/TOPSQL1", "TOPSERVER1")
If (_nCon1 < 0)
    CONOUT("Falha Conexao TOPCONN 1 - Erro: "+ str(_nCon1, 10, 0))
    return .F.
EndIf
_nCon2 := TCLink("MSSQL7/TOPSQL2", "TOPSERVER2")
If (_nCon2 < 0)
    CONOUT("Falha Conexao TOPCONN 2 - Erro: "+ str(_nCon2, 10, 0))
    return .F.
EndIf
USE CLIENTES VIA "TOPCONN" NEW // Tabela de clientes será aberto em _nCon2
TCSETCONN(_nCon1)
USE PEDIDOS VIA "TOPCONN" NEW // Tabela de pedidos será aberto em _nCon2
.
.
.
TCQUIT()
RETURN .T.
```

TCSetDummy

Altera o status do modo "dummy"

Sintaxe

TCSETDUMMY ([*IStatus*]) --> Lógico

Argumento	Obrigat.	Tipo	Descrição
<i>IStatus</i>	Sim	Lógico	Indica se deve ligar (.T.) ou desligar (.F.) o modo "dummy" .

Retorno	Descrição
.T.	Operação com sucesso
.F.	Erro na operação.

Descrição

No modo "dummy", o TopConnect faz apenas a abertura dos arquivos, sem executar nenhuma função de posicionamento. É utilizado para fazer uma abertura mais rápida dos arquivos. Passando .T., entra em modo "Dummy", .F. volta ao normal.

Exemplo

```
TCSetDummy(.t.)
For ni:= 1 to NroTabelas
    cTabela:= "TABELA"+TRIM(STR(NI, 10, 0))
    dbUseArea( .T., "TOPCONN", cTabela, cTabela, .T., .F. )
next
TCSetDummy(.F.)
dbselectarea( "TABELA1" )
DBGOTOP()
.....
```

TCSetField

Altera o tipo de um campo.

Sintaxe

TCSETFIELD (*cAlias*, *cCampo*, *cTipo* [, *nTamanho*, *nDecimais*]) --> nil

Argumento	Obrigat.	Tipo	Descrição
<i>cAlias</i>	Sim	C	Alias da Tabela
<i>cCampo</i>	Sim	C	Nome do Campo
<i>cTipo</i>	Sim	C	Novo tipo do campo. Pode ser : 'N' (numérico), 'D' (data) ou 'L' (lógico).
<i>nTamanho</i>	Não	N	Tamanho do campo. Só é utilizado se o campo for caracter ou numérico.
<i>nDecimais</i>	Não	N	Decimais do campo. Só é utilizado se o campo for numérico.

Retorno	Descrição
NIL	Sem retorno

Descrição

Quando se faz a abertura de uma tabela via TCquery, todos os campos aparecem como tipo Caracter. É necessário em seguida acertar o tipo do campo, se ele não for caracter.

Exemplo

```
aStru := CT2->(dbStruct())
```

```

cCond := ".T."
cQuery := "SELECT * FROM " + RetSqlName("CT2") + " WHERE"
cQuery := 'CT2_FILIAL = "' + xFilial("CT2") EndIf

cQuery := ChangeQuery(cQuery)
dbUseArea(.T., "TOPCONN", TCGenQry(,cQuery), 'NovoCT2', .F., .T.)

For ni := 1 to Len(aStru)
If aStru[ni,2] != 'C'
TCSetField('NovoCT2', aStru[ni,1], aStru[ni,2],aStru[ni,3],aStru[ni,4])
Endif
Next

```

TCSpExec

Executa uma Stored Procedure.

Sintaxe

TCSPEXEC (cSProc [, xParam1,...,xParamN])--> [array]

Argumento	Obrigat.	Tipo	Descrição
cSProc	Sim	C	Nome da Stored Procedure.
xParamX	Não	Qualquer	Parâmetro(s) da Stored Procedure

Retorno	Descrição
NIL	Nenhum valor é retornado pela Stored Procedure ou ocorreu um erro.
array	Array contendo os valores de retorno da Stored Procedure.

Descrição

Executa uma Stored Procedure, no banco de dados, com número variável de parâmetros.

Exemplo

A Stored Procedure abaixo retorna "Teste" e o parâmetro numérico +3:

```

Create Procedure testel( @IN_VALUE int,
                        @OUT_STR char(255), @OUT_VALUE int)
WITH RECOMPILE
As
Begin
Select @OUT_STR = "Teste", @OUT_VALUE = @IN_VALUE + 3
End
GO

```

Para executar:

```
aResult := TCSPEXEC(xProcedures (' teste1'), 100 )
IF Len(aResult) = 0
  conout("Erro na execução da Stored Procedure.")
Endif
Else
  conout(aResult[1] + str(aResult[2]))
Endif
```

TCSpExist

Verifica se uma Stored Procedure existe.

Sintaxe

TCSPEXIST (*cStoredProc*) --> Lógico

Argumento	Obrigat.	Tipo	Descrição
<i>cStoredProc</i>	Sim	C	Nome da Stored Procedure.

Retorno	Descrição
.T.	Stored Procedure existe.
.F.	Stored Procedure não existe.

Descrição

Verifica a existência de uma Stored Procedure no Banco de dados atual.

Exemplo

```
if TCSPEXist("SP000001")
  cStr := "DROP PROCEDURE " + "SP000001 "
  TCSqlExec(cStr)
endif
```

TCSqlError

Retorna o último erro produzido em comandos SQL.

Sintaxe

TCSQLERROR ()-> Caracter

Retorno	Descrição
Caracter	Texto com descrição do erro. Se não houve erro, retorna texto vazio.

Descrição

Esta função obtém as mensagens de erros em ordem inversa. Portanto, às vezes é necessário fazer várias chamadas a esta função para obter todos os erros, caso eles sejam múltiplos.

Exemplo

```
nRet = TCSQLEEXEC("INSERT INTO SALES/CUSTOMER(NAME) VALUES('JOHN DOE')")
If nRet == 0
    conout("Inserção executada")
Else
    conout("Inserção com erro (s) : ")
    cRet = TCSQLERROR()
    Do While !Empty(cRet)
        conout(cRet)
    cRet = TCSQLERROR()
    EndDo
EndIf
```

TCSrvType

Retorna o tipo de servidor.

Sintaxe

TCSRVTYPE () --> Caracter

Retorno	Descrição
""	String vazia se o driver TopConnect não estiver inicializado.
<> ""	String contendo o tipo do servidor. Pode ser: "AS/400", "WinNT", "AIX", "HPUX", "Linux"

Descrição

Retorna o tipo de servidor onde está o banco de dados.

Exemplo

```
// É utilizado para testar se o ambiente é AS/400, pois alguns comandos são diferentes
neste plataforma.
Ex: Chamada da função de reconciliação
If TcSrvType() == 'AS/400'
Processa({|lEnd| FA210Processa()})
Else
Processa({|lEnd| FA211Processa()})
Endif
```

TCSysExe

Executa um comando do sistema operacional no servidor TopConnect.

Sintaxe

TCSYSEX (*cComando*) --> Numérico

Argumento	Obrigat.	Tipo	Descrição
cComando	Sim	C	Comando do Sistema Operacional.

Retorno	Descrição
0	Comando executado com sucesso.
<>0	Erro na execução do comando.

Exemplo

```
// Copia arquivo no AS400
cAntigo:= "ANTIGO"
cNovo:= "NOVO"
cExpres := "CPYF FROMFILE("+AllTrim(cAntigo)+") TOFILE("
cExpres += AllTrim(cNovo)+") MBROPT(*ADD) FMTOPT(*MAP *DROP)"
if TCSysExe(cExpres) <> 0
conout("Erro na execução do comando")
endif
```

;TCUnLink

Finaliza uma conexão com o TopConnect.

Sintaxe

TCUNLINK([*nNroConexao*]) --> Lógico

Argumento	Obrigat.	Tipo	Descrição
<i>nNroConexao</i>	Não	N	Indica qual é o número da conexão que deve ser finalizada. Se não for passada, a conexão corrente é finalizada.

Retorno	Descrição
.T.	Execução com sucesso.
.F.	Erro. Geralmente é porque a conexão não é válida.

Descrição

Finaliza a conexão especificada com o TopConnect. Se não passar parâmetro, finaliza a conexão corrente. O número da conexão é a retornada pela função <@>TCLink.

Exemplo

```
_nCon1 := TCLink("MSSQL7/TOPSQL1", "TOPSERVER1")
_nCon2 := TCLink("MSSQL7/TOPSQL2", "TOPSERVER2")
.
.
.
TCUNLINK(_nCon1)    // Finaliza a conexão _nCon1
```

TCVUnLock

Libera o bloqueio virtual de uma string.

Sintaxe

TCVUNLOCK ([*cPalavra*]) --> Lógico

Argumento	Obrigat.	Tipo	Descrição
<i>cPalavra</i>	Não	C	String que deve ser liberada. Se não for passada, TODOS os locks virtuais desta conexão são liberados.

Retorno	Descrição
.T.	A palavra foi bloqueada.
.F.	Não foi possível desbloquear a palavra, provavelmente porque ela não estava bloqueada.

Descrição

Retira o bloqueio virtual de uma string ou de todas da conexão.

Exemplo

```
#ifdef TOP
If TCIsVLock()
if !TCVLock("Processo1")
messagebox("Nao foi possivel bloquear o processo1", "", 0)
return
endif

..... // Processa

TCVUnl ock(" Processo1")
EndIf
#endif
```

TCVLock

Bloqueia uma string através de lock virtual.

Sintaxe

TCVLOCK (*cPalavra*) --> Lógico

Argumento	Obrigat.	Tipo	Descrição
<i>cPalavra</i>	Sim	C	String que deve ser bloqueada.

Retorno	Descrição
.T.	A palavra foi bloqueada.
.F.	Não foi possível bloquear a palavra, provavelmente porque ela já foi bloqueada por outra estação.

Descrição

A função TCVLOCK bloqueia uma string através de lock virtual no servidor .

Exemplo

```
#ifdef TOP
  If TCIsVLock()
    if !TCVLock("Processo1")
      messagebox("Nao foi possivel bloquear o processo1", "", 0)
    endif
  EndIf
#endif
```

Time

Retorna a hora do sistema

Sintaxe

TIME() --> cHora

Retorno	Descrição
cHora	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.

Descrição

TIME() é uma função que retorna a hora do sistema como uma cadeia de caracteres. TIME() 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.

Exemplos

Estes exemplos mostram a função TIME() utilizada em conjunto com SUBSTR() para extrair a hora, os minutos e os segundos:

```
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
```

UnLock

Desbloqueia os registros da tabela corrente.

Sintaxe

UnLock [ALL]

Descrição

Este comando é utilizado para liberar registros da tabela corrente. Se for passada a opção "ALL" todos os registros da tabela corrente são liberados. Caso contrário, só é liberado o registro corrente.

Exemplo

```
// Este exemplo mostra como liberar todos os registros bloqueados da tabela corrente.  
DBUSEAREA( .T., "dbfcdxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F. )  
UNLOCK ALL  
// Este exemplo mostra uma variação do uso de UNLOCK para liberar apenas o registro  
corrente.  
DBUSEAREA( .T., "dbfcdxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F. )  
DBGOTO(100)  
UNLOCK // Desbloqueia o registro atual (100)
```

UpdateIntName

Atualiza o nome do índice interno da tabela CTree.

Sintaxe

UpdateIntName (*cNome*) -> Lógico

Argumento	Obrigat.	Tipo	Descrição
<i>cNome</i>	Não	C	Especifica o nome da tabela cujo índice interno deve ter o nome atualizado.

Retorno	Descrição
.F.	Não conseguiu atualizar o nome do índice interno. O arquivo não pode ser aberto em modo exclusivo.

.T.

Atualização do nome de índice interno ocorrida com sucesso

Descrição

A função UpdateIntName muda o nome do arquivo de índice interno de uma tabela Ctree, estando a mesma fechada. Para tanto ela executa os seguintes passos:

1- Abre a tabela;

2- Verifica as informações da tabela;

3- Fecha a tabela;

4- Recalcula o nome do índice interno;

5- Grava o novo nome do índice interno na tabela;

O nome do índice interno, que é armazenado no diretório "VCTREEINT\" acima da tabela equivalente, é calculado da seguinte forma:

xxxxxxxxxxxYYYMMDDhhmmss.int, sendo:

- xxxxxxxx - nome do arquivo da tabela
- xxx - extensão da tabela
- YYYMMDD - data atual
- hhmmss - horário corrente

Na próxima vez em que a tabela é aberta, o arquivo deste novo índice será criado automaticamente.

Exemplo

```
// Este exemplo demonstra o uso típico de UpdateIntName(). Se não falhar, o nome do
índice interno será atualizado e o processo continua quando ao abrir a tabela o novo
arquivo de índice interno é criado. Se falhar, uma mensagem é apresentada.
IF ! UpdateIntName("\dadosadv\sa1990.dtc")
    MessageBox("Não foi possível atualizar o nome do índice interno da tabela","Erro", 0)
    BREAK
ENDIF
USE "\dadosadv\sa1990.dtc" SHARED NEW
```

Use

Abre uma tabela na área de trabalho atual e os arquivos relacionados a ela.

Sintaxe

USE *Tabela* [INDEX *Indices*] [ALIAS *Alias*] [EXCLUSIVE | SHARED] [NEW] [READONLY] [VIA Driver]]

Argumento	Obrigat.	Tipo	Descrição
<i>Tabela</i>			Nome do arquivo da tabela a ser aberta
Indices			Nomes dos índices a serem abertos junto com a tabela
Alias			Alias da tabela a ser aberta
Driver			Nome do RDD a ser utilizado na abertura da tabela

Retorno	Descrição
NIL	Sem retorno

Descrição

Este comando associa uma tabela especificada pelo primeiro parâmetro (*Tabela*) à área de trabalho atual através de um driver especificado através do parâmetro *Driver*.

Também pode abrir os arquivos de índices relacionados com a tabela.

No caso do Ctree os arquivos de índices permanentes já são abertos automaticamente, mas nos demais RDDs deve especificar os arquivos de índices que se deseja abrir através do parâmetro *Indices*.

O Alias pode ser especificado através do parâmetro *Alias*. Pode-se especificar se a tabela será aberta em modo exclusivo ou compartilhado (através das opções "EXCLUSIV" e "SHARED").

O usuário pode optar por abrir a tabela em modo somente leitura através da opção "READONLY", onde nenhuma alteração será efetivada na tabela. A opção "NEW" determina que a tabela especificada será aberta na próxima área de trabalho disponível e será setada como a área de trabalho corrente.

Exemplo

```
// Este exemplo demonstra como utilizar o comando USE para associar uma tabela (do tipo Ctree) a próxima área de trabalho disponível e torná-la ativa em modo compartilhado e somente leitura:  
USE Cliente VIA "CTREECDX" NEW SHARED READONLY
```

Used

Verifica se existe uma tabela corrente

Sintaxe

Used() --> Lógico

Retorno	Descrição
.F.	Não existe tabela corrente
.T.	Existe tabela corrente

Descrição

Esta função é utilizada para verificar se existe alguma tabela aberta no momento.

Exemplo

```
//Este exemplo utiliza a função USED()para verificar quando a tabela está ativa:  
USE Clientes NEW  
Result := USED() // Result: .T.  
CLOSE  
Result := USED() // Result: .F.
```

Year

Converte o valor da data no valor numérico do ano.

Sintaxe

YEAR(dData) --> nAno

Argumento	Obrigat.	Tipo	Descrição
dData	Sim	D	É o valor da data a ser convertido.

Retorno	Descrição
---------	-----------

nAno	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.
0	Para uma data inválida ou nula.

Descrição

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.

Exemplos

Estes exemplos mostram YEAR() usando a data do sistema:

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

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))
```

ZAP

Remove todos os registros da tabela.

Sintaxe

ZAP

Descrição

Este comando apaga (fisicamente) todos os registro da tabela corrente.

Exemplo

```
// Este exemplo mostra como o ZAP pode se utilizado.
dbUseArea( .T., "dbfcdxads", "\dadosadv609\sa1990.dbf", "SSS", .T., .F. )
ZAP
```

Classes de Interface Visual

tSrvObject

Classe mãe de todas as classes de interface.

Características

Classe abstrata inicial de todas as classes de interface do Advpl. Não deve ser instanciada diretamente.

Propriedades

Nleft	Numérico. Coordenada horizontal em pixels.
Ntop	Numérico. Coordenada vertical em pixels.
nWidth	Numérico. Largura em pixels.
nHeight	Numérico. Altura em pixels.
cCaption	Caractere. Título ou conteúdo do objeto.
cTooltip	Caractere. Mensagem exibida quando objeto exibe seu tooltip.
IShowHint	Lógico. Flag que ativa .T. ou desativa .F. a exibição do tooltip do objeto.
cMsg	Caractere. Mensagem exibida na barra de status da janela principal quando o objeto ganha foco.
nClrText	Numérico. Cor do texto do objeto.
nClrPane	Numérico. Cor do fundo do objeto.
bWhen	Bloco de código. Executado quando há movimentação de foco na janela. Se retornar .T. o objeto continua habilitado, se retornar .F. o objeto será desabilitado.
bValid	Bloco de código. Executado quando o conteúdo do objeto é modificado e deverá ser validado. Deve retornar .T. se o conteúdo é válido e .F. se conteúdo inválido.
bIClicked	Bloco de código. Executado quando acionado click do botão esquerdo do mouse sobre o objeto.
brClicked	Bloco de código. Executado quando acionado click do botão direito do mouse sobre o objeto.
bIDbIClick	Bloco de código. Executado quando acionado duplo click do botão esquerdo do mouse sobre o objeto.
oWnd	Objeto. Janela onde o objeto foi criado.
IVisible	Booleano. Se .T. o objeto é visível, se .F. o objeto é invisível.
Cargo	Objeto ou variável. Conteúdo associado ao objeto.

bLostFocus	Bloco de código. Executado quando objeto perde foco.
bGotFocus	Bloco de código. Executado quando objeto ganha foco.

Métodos

SetFocus

Sintaxe

SetFocus()

Descrição

Força o foco de entrada de dados mudar para o objeto.

Retorno

NIL

Hide

Sintaxe

Hide()

Descrição

Torna objeto invisível.

Retorno

NIL

Show

Sintaxe

Show()

Descrição

Torna objeto visível.

Retorno

NIL

Enable

Sintaxe

Enable()

Descrição

Habilita o objeto.

Retorno

NIL

Disable

Sintaxe

Disable()

Descrição

Desabilita o objeto.

Retorno

NIL

Refresh

Sintaxe

Refresh()

Descrição

Força atualização (sincronia) de propriedades entre o programa e o Protheus Remote.

Classes de Janelas

tWindow

Classe de janela principal de programa.

Hierarquia

[tSrvObject](#) -> tWindow

Características

Classe de janela principal de programa, deverá existir apenas uma instância deste objeto na execução do programa.

Propriedades

Binit	Bloco de código. Executado quando a janela está sendo exibida.
IEscClose	Lógico. Se .T. habilita o <ESC> cancelar a execução da janela.
oCtlFocus	Objeto. Objeto contido na janela que está com foco de entrada de dados.

Métodos

New

Descrição

Método construtor da janela.

Sintaxe

New([anTop], [anLeft],[anBottom], [anRight], [acTitle], [nPar6], [oPar7], [oPar8],[oPar9], [aoParent], [IPar11], [IPar12], [anClrFore], [anClrBack], [oPar15], [cPar16], [IPar17], [IPar18], [IPar19], [IPar20], [alPixel]);

Parâmetros

AnTop	Numérico, opcional. Coordenada vertical superior em pixels ou caracteres.
AnLeft	Numérico, opcional. Coordenada horizontal esquerda em pixels ou caracteres.
AnBottom	Numérico, opcional. Coordenada vertical inferior em pixels ou caracteres.
AnRight	Numérico, opcional. Coordenada horizontal inferior em pixels ou caracteres.
acTitle	Caractere, opcional. Título da janela.
nPar6	Reservado.
oPar7	Reservado.
oPar8	Reservado.
oPar9	Reservado.
AoParent	Objeto, opcional. Janela mãe da janela corrente.
IPar11	Reservado.
IPar12	Reservado.
anClrFore	Numérico, opcional. Cor de fundo da janela.
anClrText	Numérico, opcional. Cor do texto da janela.
oPar15	Reservado.

cPar16	Reservado.
IPar17	Reservado.
IPar18	Reservado.
IPar19	Reservado.
IPar20	Reservado.
AlPixel	Lógico, opcional. Se .T. (padrão) considera coordenadas passadas em pixels, se .F. considera caracteres.

Retorno

Objeto. A janela construída.

Activate

Descrição

Ativa (exibe) a janela. Chamar esse método apenas uma vez.

Sintaxe

Activate([acShow], [bPar2], [bPar3], [bPar4], [bPar5], [bPar6], [abInit], [bPar8], [bPar9], [bPar10], [bPar11], [bPar12], [bPar13], [bPar14], [bPar15], [abValid], [bPar17], [bPar18]).

Parâmetros

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

bPar17	Reservado.
bPar18	Reservado.

Retorno

NIL

End

Descrição

Solicita encerramento da janela.

Sintaxe

End()

Retorno

Lógico. .T. se encerrou a janela e .F. se não.

Center

Descrição

Centraliza a janela.

Sintaxe

Center()

Retorno

NIL

Exemplo

```
#include "protheus.ch"

USER FUNCTION MyProg()

Local oWindow
Local abInit:= {||msgstop("ativando!")}
Local abValid:= {|| msgstop("encerrando!"),.T.}
oWindow:= tWindow():New( 10, 10, 200, 200, "Meu programa";
,,,,,,CLR_WHITE,CLR_BLACK,,,,,.T. )
oWindow:Activate( "MAXIMIZED",,,,,,abInit,,,,,,abValid,,)

Return nil
```

TDIALOG

Classe de janela de diálogo.

Hierarquia

[tSrvObject](#) -> [tWindow](#) -> tDialog

Características

Classe de janela de diálogo de entrada de dados, uso reservado, recomenda-se utilizar a classe MSDialog que é herdada desta classe.

Propriedades

Vide classes ancestrais.

Métodos

New

Descrição

Método construtor da classe.

Sintaxe

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

Parâmetros

AnTop	Numérico, opcional. Coordenada vertical superior em pixels ou caracteres.
anLeft	Numérico, opcional. Coordenada horizontal esquerda em pixels ou caracteres.
anBotom	Numérico, opcional. Coordenada vertical inferior em pixels ou caracteres.
anRight	Numérico, opcional. Coordenada horizontal direita em pixels ou caracteres.
acCaption	Caractere, opcional. Título da janela.
cPar6	Reservado.
nPar7	Reservado.
lPar8	Reservado.
nPar9	Reservado.
anClrText	Numérico,opcional. Cor do texto.

anClrBack	Numérico, opcional. Cor de fundo.
oPar12	Reservado.
AoWnd	Objeto, opcional. Janela mãe da janela a ser criada, padrão é a janela principal do programa.
AlPixel	Lógico, opcional. Se .T. considera as coordenadas passadas em pixels, se .F. considera caracteres.
oPar15	Reservado.
oPar16	Reservado.
nPar17	Reservado.
anWidth	Numérico, opcional. Largura da janela em pixels.
anHeight	Numérico, opcional. Altura da janela em pixels.

Retorno

O Diálogo criado.

Activate

Descrição

Ativa (exibe) o diálogo. Chamar somente uma vez este método.

Sintaxe

Activate([bPar1], [bPar2], [bPar3], [alCentered], [abValid], [lPar6], [abInit], [bPar8], [bPar9])

Parâmetros

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

Retorno

NIL.

End

Descrição

Encerra (fecha) o diálogo.

Sintaxe

End()

Retorno

Lógico .T. se o diálogo foi encerrado.

MSDialog

Classe de diálogo de entrada de dados.

Hierarquia

[tSrvObject](#) -> [tWindow](#) -> [tDialog](#) -> MSDialog

Características

MSDialog deve ser utilizada como padrão de janela para entrada de dados. MSDialog é um tipo de janela diálogo modal, isto é, não permite que outra janela ativa receba dados enquanto esta estiver ativa.

Propriedades

Vide classes ancestrais.

Métodos

New

Descrição

Método construtor da classe.

Sintaxe

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

Parâmetros

AnTop	Numérico, opcional. Coordenada vertical superior em pixels ou caracteres.
anLeft	Numérico, opcional. Coordenada horizontal esquerda em pixels ou caracteres.
anBottom	Numérico, opcional. Coordenada vertical inferior em pixels ou caracteres.
anRight	Numérico, opcional. Coordenada horizontal direita em pixels ou caracteres.
acCaption	Caractere, opcional. Título do diálogo.
cPar6	Reservado.
nPar7	Reservado.
lPar8	Reservado.
nPar9	Reservado.
anClrText	Cor do texto.
anClrBack	Cor do fundo.
oPar12	Reservado.
AoWnd	Objeto, opcional. Janela pai do diálogo, geralmente a janela principal do programa.
AlPixel	Se .T. considera as coordenadas informadas em pixels, se .F. considera as coordenadas em caracteres.
oPar15	Reservado.
oPar16	Reservado.
lPar17	Reservado.

Retorno

O diálogo construído.

Exemplo

```
#include "protheus.ch"

User Function Teste()

// cria diálogo
Local oDlg:=MSDialog():New(10,10,300,300,"Meu dialogo";
,,,CLR_BLACK,CLR_WHITE,,,T.)

// ativa diálogo centralizado
oDlg:Activate(,,,T.,{|msgstop("validou!"),.T.},;
,{|msgstop("iniciando...")})

Return
```


Classes Auxiliares

tFont

Classe que encapsula fonte de edição.

Hierarquia

tFontAbs -> tFont

Descrição

Utilize objeto tFont para modificar a fonte padrão de controles visuais.

Métodos

New

Descrição

Construtor do objeto.

Sintaxe

New([acName], [nPar2], [anHeight], [IPar4], [alBold], [nPar6], [IPar7], [nPar8], [allItalic], [alUnderline])

Parâmetros

acName	Caractere, opcional. Nome da fonte, o padrão é "Arial".
nPar2	Reservado.
anHeight	Numérico, opcional. Tamanho da fonte. O padrão é -11.
IPar4	Reservado.
AlBold	Lógico, opcional. Se .T. o estilo da fonte será negrito.
nPar6	Reservado.
IPar7	Reservado.
nPar8	Reservado.
allItalic	Lógico, opcional. Se .T. o estilo da fonte será itálico.

alUnderline	Lógico, opcional. Se .T. o estilo da fonte será sublinhado.
-------------	---

Retorno

O objeto criado.

Exemplo

```
#include "protheus.ch"

User Function Teste()

Local oDlg, oSay, oFont:= TFont():New("Courier New",-14,.T.)
DEFINE MSDIALOG oDlg FROM 0,0 TO 200,200 TITLE "My dialog" PIXEL
// apresenta o tSay com a fonte Courier New //
oSay:= tSay():New(10,10,{|"para exibir"},oDlg,,oFont,,
,,.T.,CLR_WHITE,CLR_RED,100,20)
oSay:lTransparent:= .T.
ACTIVATE MSDIALOG oDlg CENTERED

Return
```

Classes de Componentes

tControl

Classe abstrata mãe de todos os controles editáveis.

Hierarquia

[tSrvObject](#) -> tControl

Características

tControl é a classe comum entre todos os componentes visuais editáveis.

Propriedades

Align	Numérico. Alinhamento do controle no espaço disponibilizado pelo seu objeto parente. 0 = Nenhum (padrão), 1 = no topo, 2 = no rodapé, 3 = a esquerda, 4 = a direita e 5 = em todo o parente.
-------	--

IModified	Lógico. Se .T. indica que o conteúdo da variável associada ao controle foi modificado.
IReadOnly	Lógico. Se .T. o conteúdo da variável associada ao controle permanecerá apenas para leitura.
hParent	Numérico. Handle (identificador) do objeto sobre o qual o controle foi criado.
bChange	Bloco de código. Executado quando o estado ou conteúdo do controle é modificado pela ação sobre o controle.

Métodos

SetFocus

Descrição

Força mudança do foco de entrada de dados para o controle.

Sintaxe

SetFocus()

Retorno

NIL

Classes de Componentes Visuais

tButton

Classe de botão.

Hierarquia

[tSrvObject](#) -> [tControl](#) -> tButton

Descrição

Utilize a classe tButton para criar um controle visual do tipo botão.

Propriedades

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

Metodos

New

Descrição

Construtor da classe.

Sintaxe

New([anRow], [anCol], [acCaption], [aoWnd], [abAction], [anWidth], [anHeight], [nPar8], [aoFont], [IPar10], [aIPixel],[IPar12],[cPar13], [IPar14], [abWhen], [bPar16], [IPar17])

Parâmetros

anRow	Numérico, opcional. Coordenada vertical em pixels ou caracteres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
acCaption	Caractere, opcional. Título do botão.
aoWnd	Objeto, opcional. Janela ou controle onde o botão deverá ser criado.
abAction	Bloco de código, opcional. Bloco que deverá ser acionado quando o botão for pressionado.
anWidth	Numérico, opcional. Largura do botão em pixels.
anHeight	Numérico, opcional. Altura do botão em pixels.
nPar8	Reservado.
aoFont	Objeto, opcional. Objeto tipo tFont com propriedades da fonte utilizada para o título do botão.
IPar10	Reservado.
aIPixel	Lógico, opcional. Se .T. considera as coordenadas passadas em pixels, se .F. (padrão) considera em caracteres.
IPar12	Reservado.
cPar13	Reservado.
IPar14	Reservado.
abWhen	Bloco de código, opcional. Executado quando mudança de foco de entrada de dados está sendo efetuada na janela onde o controle foi criado. O bloco deve retornar .T. se o controle deve permanecer habilitado ou .F. se não.
bPar16	Reservado.
IPar17	Reservado.

Exemplo

```
#include "protheus.ch"

User Function TesteGet()

Local oDlg, oButton, oCombo, cCombo, aItems:= {"item1","item2","item3"}
cCombo:= aItems[2]
DEFINE MSDIALOG oDlg FROM 0,0 TO 300,300 PIXEL TITLE "Meu Combo"
oCombo:= tComboBox():New(10,10,{ |u| if(PCount()>0,cCombo:=u,cCombo) },,;
aItems,100,20,oDlg,,{ | |MsgStop("Mudou item") },,;
,,,T.,,,,,,"cCombo")
// Botão para fechar a janela
oButton:=tButton():New(30,10,"fechar",oDlg,{ | |oDlg:End() },,;
100,20,,,,,T.)
ACTIVATE MSDIALOG oDlg CENTERED
MsgStop( "O valor é "+cCombo )

Return nil
```

tCheckBox

Classe de caixa checkbox.

Hierarquia

[tSrvObject](#) -> [tControl](#) -> tCheckBox

Descrição

Utilize a classe tCheckbox quando desejar criar um controle que possua dois estados .T. ou .F..

Métodos

New

Descrição

Construtor da classe

Sintaxe

```
New([anRow], [anCol], [acCaption], [abSetGet], [aoWnd], [anWidth], [anHeight], [nPar8], [abClick],
[aoFont], [abValid], [anClrFore], [anClrBack], [IPar14], [alPixel], [cPar16], [IPar17], [abWhen])
```

Parâmetros

AnRow	Numérico, opcional. Coordenada vertical em pixels ou caracteres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.

acCaption	Caractere, opcional. Texto exibido pelo controle.
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 lógico, se <var> = .T. então o controle aparecerá checado.
aoWnd	Objeto, opcional. Janela ou controle onde o controle deverá ser criado.
anWidth	Numérico, opcional. Largura do controle em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
nPar8	Reservado.
abClick	Bloco de código, opcional. Executado quando o controle click do botão esquerdo do mouse é acionado sobre o controle.
aoFont	Objeto, opcional. Objeto tipo tFont com propriedades da fonte utilizada para o texto do controle.
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.
lPar14	Reservado.
alPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
cPar16	Reservado.
lPar17	Reservado.
abWhen	Bloco de código, opcional. Executado quando mudança de foco de entrada de dados está sendo efetuada na janela onde o controle foi criado. O bloco deve retornar .T. se o controle deve permanecer habilitado ou .F. se não.

Retorno

O objeto construído.

Exemplo

```
#include "protheus.ch"

User Function Teste()

Local oDlg, oButton, oCheck, lCheck:=.F.
DEFINE MSDIALOG oDlg FROM 0,0 TO 300,300 PIXEL TITLE "Meu programa"
oCheck:= tCheckBox():New(10,10,"funcionou?");
{|u|if( pcount()>0,lCheck:=u,lCheck)};
,oDlg,100,20,,,,,,.T.)
oButton:=tButton():New(30,10,"fechar",oDlg,{||oDlg:End()});
```

```
100,20,,,,,T.)  
ACTIVATE MSDIALOG oDlg CENTERED  
If lCheck  
MsgStop( "Funcionou!" )  
endif  
  
Return nil
```

tComboBox

Classe de combobox.

Hierarquia

[tSrvObject](#) -> [tControl](#) -> tComboBox

Descrição

Utilize a classe tComboBox para criar uma entrada de dados com múltipla escolha com item definido em uma lista vertical, acionada por F4 ou pelo botão esquerdo localizado na parte direita do controle. A variável associada ao controle terá o valor de um dos itens selecionados ou no caso de uma lista indexada, o valor de seu índice.

Propriedades

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

Métodos

New

Descrição

Construtor da classe.

Sintaxe

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

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. Se a lista for seqüencial, o controle atualizará <var> com o conteúdo do item selecionado, se a lista for indexada, <var> será atualizada com o valor do índice do item selecionado.
anItems	Array, opcional. Lista de itens, caracteres, a serem exibidos. Pode ter os seguintes formatos: a) Seqüencial, exemplo: { "item1", "item2", ..., "itemN" } ou b) Indexada, exemplo: { "a=item1", "b=item2", ..., "n=itemN" }.
anWidth	Numérico, opcional. Largura do controle em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
aoWnd	Objeto, opcional. Janela ou controle onde o controle será criado.
nPar8	Reservado.
abChange	Bloco de código, opcional. Executado quando o controle modifica o item selecionado.
abValid	Bloco de código, opcional. Executado quando o conteúdo do controle deve ser validado, deve retornar .T. se o conteúdo for válido e .F. quando o conteúdo for inválido.
anClrBack	Numérico, opcional. Cor de fundo do controle.
anClrText	Numérico, opcional. Cor do texto do controle.
alPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
aoFont	Objeto, opcional. Objeto tipo tFont utilizado para definir as características da fonte utilizada para exibir o conteúdo do controle.
cPar15	Reservado.
lPar16	Reservado.
abWhen	Bloco de código, opcional. Executado quando mudança de foco de entrada de dados está sendo efetuada na janela onde o controle foi criado. O bloco deve retornar .T. se o controle deve permanecer habilitado ou .F. se não.
lPar18	Reservado.
aPar19	Reservado.
bPar20	Reservado.
cPar21	Reservado.
acReadVar	Caractere, opcional. Nome da variável que o controle deverá manipular, deverá ser a mesma variável informada no parâmetro abSetGet, e será o retorno da função ReadVar().

Retorno

O objeto criado.

Select

Descrição

Muda o item selecionado no combobox.

Sintaxe

Select([anItem])

Parâmetros

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

Retorno

NIL

Exemplo

```
#include "protheus.ch"

User Function TesteGet()

Local oDlg, oButton, oCombo, cCombo, aItems:= {"item1","item2","item3"}
cCombo:= aItems[2]
DEFINE MSDIALOG oDlg FROM 0,0 TO 300,300 PIXEL TITLE "Meu Combo"
oCombo:= tComboBox():New(10,10,{ |u| if(PCount()>0,cCombo:=u,cCombo) }, ;
aItems,100,20,oDlg,,{ | |MsgStop("Mudou item") }, ;
,,,T,,,,,,,,,"cCombo")
// Botão para fechar a janela
@ 40,10 BUTTON oButton PROMPT "Fechar" OF oDlg PIXEL ACTION oDlg:End()
ACTIVATE MSDIALOG oDlg CENTERED
MsgStop( "O valor é "+cCombo )

Return nil
```

tGet

Classe de controle para entrada de dados editáveis.

Hierarquia

[tSrvObject](#) -> [tControl](#) -> tGet

Descrição

Use tGet 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.

Propriedades

IPassword	Lógico. Se .T. o controle se comporta como entrada de dados de senha, exibindo asteriscos "*" para esconder o conteúdo digitado.
Picture	Caractere. Máscara de formatação do conteúdo a ser exibido.

Métodos

New

Descrição

Método construtor do controle.

Sintaxe

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.

Retorno

O controle construído.

Exemplo

```
#include "protheus.ch"

User Function TesteGet()

Local oDlg, oGet1, oButton, nGet1:=0
DEFINE MSDIALOG oDlg FROM 0,0 TO 300,300 PIXEL TITLE "Meu Get"
oGet1:= TGet():New(10,10,{|u| if(PCount())>0,nGet1:=u,nGet1}}, oDlg,;
100,20,"@E 999,999.99",;
{|o|nGet1>1000.00},,,,,,T,,,,,,,,,"nGet1")
/* Tem o mesmo efeito
@ 10,10 MSGET oGet1 VAR nGet1 SIZE 100,20 OF oDlg PIXEL PICTURE "@E 999,999.99" VALID
nGet1>1000.00
*/
// Botão para fechar a janela
@ 40,10 BUTTON oButton PROMPT "Fechar" OF oDlg PIXEL ACTION oDlg:End()
ACTIVATE MSDIALOG oDlg CENTERED
MsgStop( "O valor é "+Transform(nGet1,"@E 999,999.00") )

Return nil
```

tGroup

Classe de painel de grupo de controles.

Hierarquia

[tSrvObject](#) -> [tControl](#) -> tGroup

Descrição

Utilize a classe tGroup para criar um painel onde controles visuais podem ser agrupados ou classificados. É criada uma borda com título em volta dos controles agrupados.

Métodos

New

Descrição

Construtor da classe.

Sintaxe

New([anTop], [anLeft], [anBottom], [anRight], [acCaption], [aoWnd], [anClrText], [anClrPane], [alPixel], [IPar10])

Parâmetros

anTop	Numérico, opcional. Coordenada vertical superior em pixels ou caracteres.
anLeft	Numérico, opcional. Coordenada horizontal esquerda em pixels ou caracteres.
anBottom	Numérico, opcional. Coordenada vertical inferior em pixels ou caracteres.
anRight	Numérico, opcional. Coordenada horizontal direita em pixels ou caracteres.
acCaption	Caractere, opcional. Título do grupo.
aoWnd	Objeto, opcional. Janela ou controle onde o controle será criado.
anClrText	Numérico, opcional. Cor do texto.
anClrPane	Numérico, opcional. Cor do fundo.
alPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
IPar10	Reservado.

Retorno

O objeto criado.

Exemplo

```
#include "protheus.ch"

User function teste()
```

```
Local oDlg, oGroup, oGet1, oGet2, cGet1:=Space(10),;
cGet2:= Space(10)
DEFINE MSDIALOG oDlg FROM 0,0 TO 400,400 TITLE "My test" PIXEL
oGroup:= tGroup():New(10,10,200,200,"grupo de gets",oDlg,,,T.)
@ 10,10 MSGET oGet1 VAR cGet1 SIZE 100,10 OF oGroup PIXEL
@ 30,10 MSGET oGet2 VAR cGet2 SIZE 100,10 OF oGroup PIXEL
ACTIVATE MSDIALOG oDlg CENTERED

Return nil
```

tListbox

Classe de lista de itens.

Hierarquia

[tSrvObject](#) -> [tControl](#) -> tListbox

Descrição

Utilize a classe tListbox para criar uma janela com itens selecionáveis e barra de rolagem. Ao selecionar um item, uma variável é atualizada com o conteúdo do item selecionado.

Parâmetros

NAt	Numérico. Indica ou modifica o item selecionado.
altems	Array de itens caracteres. Lista do itens selecionáveis.

Métodos

New

Descrição

Contrutor da classe

Sintaxe

```
New([anRow], [anCol], [abSetGet], [aaltems], [anWidth], [anHeigth], [abChange], [aoWnd], [abValid],
[anClrFore], [anClrBack], [alPixel], [IPar13], [abLDBLClick], [aoFont], [cPar16], [IPar17], [abWhen],
[aPar19], [bPar20], [IPar21], [IPar22], [abRightClick] )
```

Parâmetros

anRow	Numérico, opcional. Coordenada vertical em pixels ou caracteres.
-------	--

anCol	Númerico, opcional. Coordenada horizontal em pixels ou caracteres.
abSetGet	Bloco de código, opcional. Bloco de código no formato { u if(Pcount()>0, <var>:= u, <var>)} que o controle utiliza para atualizar a variável <var>. <var> deve ser tipo caracter ou numérica.
aaltems	Array de itens caracteres, opcional. Lista de itens selecionáveis.
anWidth	Númerico, opcional. Largura do controle em pixels.
anHeight	Númerico, opcional. Altura do controle em pixels.
abChange	Bloco de código, opcional. Executado quando o item selecionado é alterado.
aoWnd	Objeto, opcional. Janela ou controle onde o controle será criado.
abValid	Bloco de código, opcional. Executado quando o conteúdo do controle deve ser validado, deve retornar .T. se o conteúdo for válido e .F. quando o conteúdo for inválido.
anClrFore	Númerico, opcional. Cor de fundo do controle.
anClrBack	Númerico, opcional. Cor do texto do controle.
alPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
IPar13	Reservado.
abLDBLClick	Bloco de código, opcional. Executado quando acionado duplo click do botão esquerdo do mouse sobre o controle.
aoFont	Objeto, opcional. Objeto tipo tFont utilizado para definir as características da fonte utilizada para exibir o conteúdo do controle.
cPar16	Reservado.
IPar17	Reservado.
abWhen	Bloco de código, opcional. Executado quando mudança de foco de entrada de dados está sendo efetuada na janela onde o controle foi criado. O bloco deve retornar .T. se o controle deve permanecer habilitado ou .F. se não.
aPar19	Reservado.
bPar20	Reservado.
IPar21	Reservado.
IPar22	Reservado.
abRightClick	Bloco de código, opcional. Executado quando acionado click do botão direito do mouse sobre o controle.

Select

Descrição

Força a seleção de um item.

Sintaxe

Select([nItem])

Parâmetros

nItem	Numérico, opcional. Item a ser selecionado.
-------	---

Retorno

NIL

Add

Descrição

Insere ou adiciona novo item.

Sintaxe

Add(cText, nPos)

Parâmetros

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

Retorno

NIL

Modify

Descrição

Modifica o texto de um item.

Sintaxe

Modify(cText, nPos)

Parâmetros

cText	Caractere, obrigatório. Texto novo.
nPos	Numérico, obrigatório. Posição a ser modificada deve ser maior que 0 e menor ou igual que o número de itens.

Retorno

NIL

Del

Descrição

Apaga um item.

Sintaxe

Del(nPos)

Parâmetros

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

Retorno

NIL

Len

Descrição

Retorna o número de itens.

Sintaxe

Len()

Retorno

Numérico. Número de itens.

Reset

Descrição

Apaga todos os itens.

Sintaxe

Reset()

Retorno

NIL

Exemplo

```
#include "protheus.ch"

User Function Teste()
Local oDlg, oList, nList:= 1, aItems:={}
Add(aItems,"Item 1")
Add(aItems,"Item 2")
Add(aItems,"Item 3")
Add(aItems,"Item 4")
DEFINE MSDIALOG oDlg FROM 0,0 TO 400,400 PIXEL TITLE "Teste"
oList:= tListBox():New(10,10,{ |u| if(Pcount(>)>0,nList:=u,nList)});
```



```
,aItems,100,100,,oDlg,,,,.T.)  
ACTIVATE MSDIALOG oDlg CENTERED  
  
Return nil
```

tMeter

Classe de régua de processamento.

Hierarquia

[tSrvObject](#) -> [tControl](#) -> tMeter

Descrição

Utilize a classe tMeter para criar um controle que exibe uma régua (gauge) de processamento, descrevendo o andamento de um processo através da exibição de uma barra horizontal.

Parâmetros

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

Métodos

New

Descrição

Contrutor da classe.

Sintaxe

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

Parâmetros

anRow	Numérico, opcional. Coordenada vertical em pixels ou caracteres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.

abSetGet	Bloco de código, opcional. Bloco de código no formato { u if(Pcount()>0, <var>:= u, <var>) } que o controle utiliza para atualizar a variável <var>. <var> deve ser tipo numérico.
anTotal	Numérico, opcional. Numero total de passos até o preenchimento da régua de processo.
aoWnd	Objeto, opcional. Janela ou controle onde o controle sera criado.
anWidth	Numérico, opcional. Largura do controle em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
lPar8	Reservado.
alPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
oPar10	Reservado.
cPar11	Reservado.
alNoPerc	Lógico, opcional. Se .T. (padrão) não considera os passos de atualização em porcentagem.
anClrPane	Numérico, opcional. Cor de fundo do controle.
nPar14	Reservado.
anClrBar	Numérico, opcional. Cor da barra de andamento.
nPar16	Reservado.
lPar17	Reservado.

Retorno

O objeto construído.

Set

Descrição

Atualiza a posição da régua de processamento.

Sintaxe

Set([nVal])

Parâmetros

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

Retorno

NIL

Exemplo

```
#include "protheus.ch"

Static lRunning:=.F., lStop:=.F.

User Function Teste()

Local oDlg, oMeter, nMeter:=0, oBtn1, oBtn2
DEFINE MSDIALOG oDlg FROM 0,0 TO 400,400 TITLE "Teste"
// cria a régua
oMeter:= tMeter():New(10,10,{|u|if(Pcount())>0,nMeter:=u,nMeter});
,100,oDlg,100,20,,.T.)
// botão para ativar andamento da régua
@ 30,10 BUTTON oBtn1 PROMPT "Run" OF oDlg PIXEL ACTION RunMeter(oMeter)
@ 50,10 BUTTON oBtn2 PROMPT "Stop" OF oDlg PIXEL ACTION lStop:=.T.
ACTIVATE MSDIALOG oDlg CENTERED

Return nil

Static Function RunMeter(oMeter)

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

Return
```

tMultiget

Classe de campo Memo de edição.

Hierarquia

[tSrvObject](#) -> [tControl](#) -> tMultiGet

Descrição

Utilize a classe tMultiget para criar controle de edição de texto de múltiplas linhas.

Propriedades

IWordWrap	Lógico. Se .T., faz quebra automática de linhas.
-----------	--

Métodos

New

Descrição

Construtor da classe.

Sintaxe

New([anRow], [anCol], [abSetGet], [aoWnd], [anWidth], [anHeight], [aoFont], [alHScroll], [anClrFore], [anClrBack], [oPar11], [alPixel], [cPar13], [IPar14], [abWhen], [IPar16], [IPar17], [alReadOnly], [abValid], [bPar20], [IPar21], [alNoBorder], [alNoVScroll])

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.
aoWnd	Objeto, opcional. Janela ou controle onde o controle será criado.
anWidth	Numérico, opcional. Largura do controle em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
aoFont	Objeto, opcional. Objeto tipo tFont utilizado para definir as características da fonte utilizada para exibir o conteúdo do controle.
alHScroll	Lógico, opcional. Se .T., habilita barra de rolagem horizontal.
anClrFore	Numérico, opcional. Cor de fundo do controle.
anClrBack	Numérico, opcional. Cor do texto do controle.
oPar11	Reservado.
alPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.
cPar13	Reservado.
IPar14	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.
IPar16	Reservado.
IPar17	Reservado.

alReadOnly	Lógico, opcional. Se .T. o controle so permitira leitura.
abValid	Bloco de código, opcional. Executado quando o conteúdo do controle deve ser validado, deve retornar .T. se o conteúdo for válido e .F. quando o conteúdo for inválido.
bPar20	Reservado.
lPar21	Reservado.
alNoBorder	Lógico, opcional. Se .T. cria controle sem borda.
alNoVScroll	Lógico, opcional. Se .T., habilita barra de rolagem vertical.

Retorno

O objeto construído.

EnableVScroll

Descrição

Habilita a barra de rolagem vertical.

Sintaxe

EnableVScroll(IEnable)

Parâmetros

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

Retorno

NIL

EnableHScroll

Descrição

Habilita a barra de rolagem horizontal.

Sintaxe

EnableHScroll(IEnable)

Parâmetros

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

Retorno

NIL

Exemplo

```
#include "protheus.ch"

User Function Teste()
Local oDlg, oMemo, cMemo:= space(50)
DEFINE MSDIALOG oDlg FROM 0,0 TO 400,400 PIXEL TITLE "My test"
oMemo:= tMultiget():New(10,10,{|u|if(Pcount(>)>0,cMemo:=u,cMemo)};
,oDlg,100,100,,,,,T.)
@ 200,10 BUTTON oBtn PROMPT "Fecha" OF oDlg PIXEL ACTION oDlg:End()
ACTIVATE MSDIALOG oDlg CENTERED
MsgStop(cMemo)

Return Nil
```

tPanel

Classe de painel estático.

Hierarquia

[tSrvObject](#) -> [tControl](#) -> tPanel

Descrição

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

Métodos

New

Descrição

Construtor da classe.

Sintaxe

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

Parâmetros

anRow	Númerico, opcional. Coordenada vertical em pixels.
anCol	Númerico, opcional. Coordenada horizontal em pixels.
acText	Caractere, opcional. Texto a ser exibido ao fundo.

aoWnd	Objeto, opcional. Janela ou controle onde será criado o objeto.
alCentered	Lógico, opcional. Se .T. exibe o texto de título ao centro do controle.
IPar6	Reservado.
anClrText	Numérico, opcional. Cor do texto do controle.
anClrBack	Numérico, opcional. Cor do fundo do controle.
anWidth	Numérico, opcional. Largura do controle em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
alLowered	Lógico, opcional. Se .T. exibe o painel rebaixado em relação ao controle de fundo.
alRaised	Lógico, opcional. Se .T. exibe a borda do controle rebaixada em relação ao controle de fundo.

Retorno

O objeto construído.

Exemplo

```
#include "protheus.ch"

User Function Teste()

Local oDlg, oPanel, oBtn1, oBtn2
DEFINE MSDIALOG oDlg FROM 0,0 TO 400,400 PIXEL TITLE "My test"
// cria o painel
oPanel:= tPanel():New(10,10,"",oDlg,,,,CLR_BLUE,100,100)
// cria botão sobre o painel
@ 10,10 BUTTON oBtn1 PROMPT "hide" OF oPanel ACTION oPanel:Hide()
// cria botão fora o painel
@ 200,10 BUTTON oBtn2 PROMPT "show" OF oDlg ACTION oPanel:Show()
ACTIVATE MSDIALOG oDlg CENTERED

Return
```

tRadMenu

Classe de radio group.

Hierarquia

[tSrvObject](#) -> [tControl](#) -> tRadMenu

Descrição

Utilize a classe tRadMenu para criar um controle que possibilita escolha de item através de uma lista.

Propriedades

nOption	Numérico. Item selecionado.
alItems	Array de caracteres. Lista de itens selecionáveis.

Métodos

New

Descrição

Construtor do objeto.

Sintaxe

New([anRow], [anCol], [aacItems], [abSetGet], [aoWnd], [aPar6], [abChange], [anClrText], [anClrPan], [cPar10], [IPar11], [abWhen], [anWidth], [anHeight], [abValid], [IPar16], [IPar17], [alPixel])

Parâmetros

anRow	Numérico, opcional. Coordenada vertical em pixels ou caracteres.
anCol	Numérico, opcional. Coordenada horizontal em pixels ou caracteres.
aacItems	Array de caracteres, opcional. Lista de opções.
abSetGet	Bloco de código, opcional. Bloco de código no formato { u if(Pcount()>0, <var>:= u, <var>) } que o controle utiliza para atualizar a variável <var>. <var> deve ser tipo numérico.
aoWnd	Objeto, opcional. Janela ou controle onde o controle será criado.
aPar6	Reservado.
abChange	Bloco de código, opcional. Executado quando o item selecionado é alterado.
anClrText	Numérico, opcional. Cor do texto do controle
anClrPan	Numérico, opcional. Cor de fundo do controle.
cPar10	Reservado.
IPar11	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. para que o controle permaneça habilitado, ou .F. se não.
anWidth	Numérico, opcional. Largura do controle em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
abValid	Bloco de código, opcional. Executado quando o conteúdo do controle deva ser validado, retornando .T. se o conteúdo for válido, e .F. quando inválido.
IPar16	Reservado.

Lpar17	Reservado.
alPixel	Lógico, opcional. Se .T. as coordenadas informadas são em pixels, se .F. são em caracteres.

Retorno

O objeto construído.

EnableItem

Descrição

Habilita ou desabilita item.

Sintaxe

EnableItem([nItem], [IEnable])

Parâmetros

nItem	Numérico, opcional. Item selecionado.
IEnable	Lógico, opcional. Se .T. habilita o item se .F. desabilita o item.

Retorno

NIL

Exemplo

```
#include "protheus.ch"

User Function Teste()
Local oDlg, oButton, oRadio, nRadio:=1,
aOptions:={"escolha1","escolha2"}
DEFINE MSDIALOG oDlg FROM 0,0 TO 300,300 PIXEL TITLE "Meu Get"
oRadio:= tRadMenu():New(10,10,aOptions,
{|u|if(PCount(>0,nRadio:=u,nRadio)},,
oDlg,,,,,,,,100,20,,,,.T.)
@ 40,10 BUTTON oButton PROMPT "7Fechar" OF oDlg PIXEL ACTION oDlg:End()
ACTIVATE MSDIALOG oDlg CENTERED
MsgStop("Escolheu "+aOptions[nRadio] )

Return nil
```

tSay

Classe de label.

Hierarquia

[tSrvObject](#) -> [tControl](#) -> tSay

Descrição

O objeto tipo tSay exibe o conteúdo de texto estático sobre uma janela ou controle.

Parâmetros

IWordWrap	Lógico. Se .T. quebra o texto em várias linhas de maneira a enquadrar o conteúdo na área determinada para o controle, sendo o padrão .F.
ITransparent	Lógico. Se .T. a cor de fundo do controle é ignorada assumindo o conteúdo ou cor do controle ou janela ao fundo, sendo o padrão .T.

Métodos

New

Descrição

Método construtor do controle.

Sintaxe

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	Codeblock, opcional. Quando executado deve retornar uma cadeia de caracteres a ser exibida.
aoWnd	Objeto, opcional. Janela ou diálogo onde o controle será criado.
acPicture	Caractere, opcional. Picture de formatação do conteúdo a ser exibido.
aoFont	Objeto, opcional. Objeto tipo tFont para configuração do tipo de fonte que será utilizado para exibir o conteúdo.
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.

Retorno

O controle criado.

SetText

Descrição

Modifica o conteúdo a ser exibido pelo controle.

Sintaxe

SetText([xVal])

Parâmetros

xVal	Caracter / Numérico / Data, Opcional. Valor a ser exibido.
------	--

Retorno

NIL

Exemplo

```
#include "protheus.ch"

User Function Teste()

Local oDlg, oSay
DEFINE MSDIALOG oDlg FROM 0,0 TO 200,200 TITLE "My dialog" PIXEL
oSay:= tSay():New(10,10,{||"para exibir"},oDlg,,,;
,,.T.,CLR_WHITE,CLR_RED,100,20)
ACTIVATE MSDIALOG oDlg CENTERED

Return
```

tScrollbar

Classe de área de scroll.

Hierarquia

[tSrvObject](#) -> [tControl](#) -> tScrollbar

Descrição

Utilize a classe tScrollbar para criar um painel com scroll deslizantes nas laterais do controle.

Métodos

New

Descrição

Construtor da classe

Sintaxe

New([aoWnd], [anTop], [anLeft], [anHeight], [anWidth], [alVertical], [alHorizontal], [alBorder])

Parâmetros

aoWnd	Objeto, opcional. Janela ou controle onde o controle será criado.
anTop	Numérico, opcional. Coordenada vertical em pixels.
anLeft	Numérico, opcional. Coordenada horizontal em pixels.
anHeight	Numérico, opcional. Altura do controle em pixels.
anWidth	Numérico, opcional. Largura do controle em pixels.
alVertical	Lógico, opcional. Se .T. exibe a barra de scroll vertical.
alHorizontal	Lógico, opcional. Se .T. exibe a barra de scroll horizontal.
alBorder	Lógico, opcional. Se .T. exibe a borda do controle.

Retorno

O objeto criado.

Exemplo

```
#include "protheus.ch"

User Function Teste()

Local oDlg, oScr, oGet1, oGet2, oGet3
Local cGet1, cGet2, cGet3
cGet1:= Space(10)
cGet2:= Space(10)
cGet3:= Space(10)
DEFINE MSDIALOG oDlg FROM 0,0 TO 400,400 PIXEL "My test"
oScr:= TScrollBar():New(oDlg,10,10,200,200,.T.,.T.,.T.)
// cria controles dentro do scrollbar
@ 10,10 MSGEST oGet1 VAR cGet1 SIZE 100,10 OF oScr PIXEL
@ 50,10 MSGEST oGet2 VAR cGet2 SIZE 100,10 OF oScr PIXEL
@ 150,100 MSGEST oGet3 VAR cGet3 SIZE 100,10 OF oScr PIXEL
ACTIVATE MSDIALOG oDlg CENTERED

Return nil
```

Infra-estrutura

Bem-vindo à documentação das funções genéricas e objetos disponíveis no RPO padrão do Advanced Protheus.

Esta documentação aborda os seguintes tópicos:

- [Funções de Infra-Estrutura](#)
- [Objetos de Infra- Estrutura](#)

MsGetDados

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

Características

A MsGetDados precisa que sejam declaradas as variáveis abaixo sendo tipo Private:

aRotina.

Vetor com as rotinas que serão executadas na MBrowse e que definira o tipo de operação que esta sendo executada (inclusão, alteração, exclusão, visualização, pesquisa, ...) no formato:

Elemento	Conteúdo
1	Título
2	Rotina
3	Reservado
4	Operação (1 - pesquisa; 2 - visualização; 3 - inclusão; 4 - alteração; 5 - exclusão)
5	Acesso relacionado a rotina, se esta posição não for informada nenhum acesso será validado

aHeader

Vetor com informações das colunas no formato:

Elemento	Conteúdo
1	Título
2	Campo
3	Máscara
4	Tamanho
5	Decimal
6	Validação
7	Reservado
8	Tipo
9	Reservado
10	Reservado

aCols

Vetor com as linhas a serem editadas. As colunas devem ser construídas com base no aHeader mais uma última com valor lógico que determina se a linha foi excluída.

lRefresh

Variável tipo lógica para uso reservado.

A MsGetDados cria a variável publica n que indica qual a linha posicionada do aCols.

As funções passadas como parâmetro para a MsGetDados (cLinhaOk, cTudoOk, ...) não poderão ser declaradas como Static Function.

A consulta padrão, validação do usuário e gatilhos estarão habilitados se o campo estiver cadastrado no Dicionário de Dados (SX3/SX7) e apresentar estas opções disponíveis.

Métodos

New

Descrição

Cria o objeto MsGetDados.

Sintaxe

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

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

Retorno

Objeto.

ForceRefresh

Descrição

Atualiza a MsGetDados com o aCols e posiciona na primeira linha.

Sintaxe

oObjeto: ForceRefresh()

Hide

Descrição

Esconde a MsGetDados.

Sintaxe

oObjeto: Hide()

Show

Descrição

Mostra a MsGetDados.

Sintaxe

oObjeto: Show()

Exemplo

```
User Function Exemplo()  
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", "AxAlterar", 0, 4},;  
                    { "Excluir", "AxDeleta", 0, 5}}  
  
DbSelectArea("SX3")  
DbSetOrder(1)  
DbSeek("SA1")  
While !Eof() .and. SX3->X3_ARQUIVO == "SA1"  
    If X3Uso(SX3->X3_USADO) .and. cNivel >= SX3->X3_NIVEL  
        nUsado++  
        Add(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.
    DEFINE MSDIALOG oDlg TITLE "Exemplo" FROM 00,00 TO 300,400 PIXEL
    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)
    ACTIVATE MSDIALOG oDlg CENTERED
    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.

```

MsGetDb

Objeto tipo lista com uma ou mais colunas para cadastramento de dados baseado em uma tabela temporária.

Características

A MsGetDB precisa que sejam declaradas as variáveis abaixo sendo tipo Private:

aRotina.

Vetor com as rotinas que serão executadas na MBrowse e que definira o tipo de operação que esta sendo executada (inclusão, alteração, exclusão, visualização, pesquisa, ...) no formato:

Elemento	Conteúdo
1	Título
2	Rotina
3	Reservado
4	Operação (1 - pesquisa; 2 - visualização; 3 - inclusão; 4 - alteração; 5 - exclusão)

5

Acesso relacionado a rotina, se esta posição não for informada nenhum acesso será validado

aHeader

Vetor com informações das colunas no formato:

Elemento	Conteúdo
1	Título
2	Campo
3	Máscara
4	Tamanho
5	Decimal
6	Validação
7	Reservado
8	Tipo
9	Reservado
10	Reservado

IRefresh

Variável tipo lógica para uso reservado.

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.

A MsGetDB cria a variável publica nBrLin que indica qual a linha posicionada do aCols.

As funções passadas como parâmetro para a MsGetDB (cLinhaOk, cTudoOk, ...) não poderão ser declaradas como Static Function.

A consulta padrão, validação do usuário e gatilhos estarão habilitados se o campo estiver cadastrado no Dicionário de Dados (SX3/SX7) e apresentar estas opções disponíveis.

Métodos

New

Descrição

Cria o objeto MsGetDB.

Sintaxe

```
MSGETDB():NEW( nSuperior, nEsquerda, nInferior, nDireita, nOpc, [ cLinhaOk ], [ cTudoOk ], [ cIniCpos ], [ lApagar ], [ aAlter ], [ nCongelar ], [ lVazio ], [ uPar1 ], cTRB, [ cCampoOk ], [ lCondicional ], [ lAdicionar ], [ oWnd ], [ lDisparos ], [ uPar2 ], [ cApagarOk ], [ cSuperApagar ] ) -> objeto
```

Argumentos	Descrição
<i>nSuperior</i>	Distancia entre a MsGetDB e o extremidade superior do objeto que a contém.
<i>nEsquerda</i>	Distancia entre a MsGetDB e o extremidade esquerda do objeto que a contém.
<i>nInferior</i>	Distancia entre a MsGetDB e o extremidade inferior do objeto que a contém.
<i>nDireita</i>	Distancia entre a MsGetDB e o extremidade direita do objeto que a contém.
<i>nOpc</i>	Posição do elemento do vetor aRotina que a MsGetDB usará como referência.
<i>cLinhaOk</i>	Função executada para validar o contexto da linha atual do aCols.
<i>cTudoOk</i>	Função executada para validar o contexto geral da MsGetDB (todo aCols).
<i>cIniCpos</i>	Nome dos campos do tipo caracter que utilizarão incremento automático. Este parâmetro deve ser no formato “+<nome do primeiro campo>+<nome do segundo campo>+...”.
<i>lApagar</i>	Habilita a opção de deletar linhas do aCols. Valor padrão falso.
<i>aAlter</i>	Vetor com os campos que poderão ser alterados.
<i>nCongelar</i>	Indica qual coluna não ficara congelada na exibição.
<i>lVazio</i>	Habilita validação da primeira coluna do aCols para esta não poder estar vazia. Valor padrão falso.
<i>uPar1</i>	Parâmetro reservado.
<i>cTRB</i>	Alias da tabela temporária.
<i>cCampoOk</i>	Função executada na validação do campo.
<i>lConditional</i>	Reservado
<i>lAdacionar</i>	Indica se a MsGetDB ira criar uma linha em branco automaticamente quando for inclusão.
<i>oWnd</i>	Objeto no qual a MsGetDB será criada.
<i>lDisparos</i>	Indica se será utilizado o Dicionário de Dados para consulta padrão, inicialização padrão e gatilhos.
<i>uPar2</i>	Parâmetro reservado.
<i>cApagarOk</i>	Função executada para validar a exclusão de uma linha do aCols.
<i>cSuperApagar</i>	-Função executada quando pressionada as teclas <Ctrl>+<Delete>.

Retorno

Objeto.

ForceRefresh

Descrição

Atualiza a MsGetDB com a tabela e posiciona na primeira linha.

Sintaxe

ForceRefresh()

Exemplo

```
User Function Exemplo()
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")
While !Eof() .and. SX3->X3_ARQUIVO == "SA1"
If X3Uso(SX3->X3_USADO) .and. cNivel >= SX3->X3_NIVEL
    nUsado++
    Add(aHeader,{Trim(X3Titulo()),;
                SX3->X3_CAMPO,;
                SX3->X3_PICTURE,;
                SX3->X3_TAMANHO,;
                SX3->X3_DECIMAL,;
                SX3->X3_VALID,;
                "",;
                SX3->X3_TIPO,;
                "",;
                "" })
    Add(aStruct,{SX3->X3_CAMPO,;
                SX3->X3_TIPO,;
                SX3->X3_TAMANHO,;
                SX3->X3_DECIMAL})

EndIf
DbSkip()
End
Add(aStruct,{ "FLAG", "L", 1, 0})
cCriaTrab := CriaTrab(aStruct,.T.)
DbUseArea(.T., __LocalDriver, cCriaTrab,,.T.,.F.)
DEFINE MSDIALOG oDlg TITLE "Exemplo" FROM 00,00 TO 300,400 PIXEL
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" )
ACTIVATE MSDIALOG oDlg CENTERED
DbSelectArea(cCriaTrab)
DbCloseArea()
Return

User Function LINHAOK()
ApmMsgStop("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.

```

MsmGet

Conjunto de objetos tipo MsGet/MsCombo para cadastramento de dados baseado no Dicionário de Dados.

Características

A MsMGet também poder criada a partir da função Enchoice a qual recebe os mesmos parâmetros do método New porém não retorna o objeto criado.

A MsMGet se baseia no Dicionário de Dados para verificar campos obrigatórios, validações, gatilhos, consulta padrão etc. assim como para a criação das Pastas de Cadastro.

A MsMGet pode usar tanto variáveis de memória do tipo Private como diretamente os campos da tabela a que se refere.

Métodos

New

Descrição

Cria o objeto MsMGet.

Sintaxe

```

MSMGET():NEW( cAlias, nReg, nOpc, [ uPar1 ], [ uPar2 ], [ uPar3 ], [ aAcho ], [ aPos ], [ aCpos ], [
uPar4 ], [ uPar5 ], [ uPar6 ], [ uPar7 ], [ oWnd ], [ uPar8 ], [ lMemoria ], [ lColuna ], [ uPar9 ], [
lSemPastas ] ) -> objeto

```

Argumentos	Descrição
<i>cAlias</i>	Alias do dados a serem cadastrados.
<i>nReg</i>	Número do registro da tabela a ser editado.

<i>uPar1</i>	Parâmetro reservado.
<i>uPar2</i>	Parâmetro reservado.
<i>uPar3</i>	Parâmetro reservado.
<i>aAcho</i>	Vetor com os campos que serão apresentados pela MsMGet.
<i>aPos</i>	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.
<i>aCpos</i>	Vetor com os campos que poderão ser alterados.
<i>uPar4</i>	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>+...”.
<i>uPar5</i>	Parâmetro reservado.
<i>uPar6</i>	Parâmetro reservado.
<i>uPar7</i>	Parâmetro reservado.
<i>oWnd</i>	Objeto no qual a MsMGet será criada.
<i>uPar8</i>	Parâmetro reservado.
<i>IMemoria</i>	Indica se será usado variáveis de memória ou os campos da tabela para cadastramento dos dados. Valor padrão falso.
<i>IColuna</i>	Indica se a MsMGet sera apresentada com um objeto por linha (uma coluna). Valor padrão falso. Parâmetro reservado.
<i>uPar9</i>	Parâmetro reservado.
<i>ISemPastas</i>	Indica se não será usado as Pastas de Cadastro na MsMGet. Função executada para validar a exclusão de uma linha do aCols.

Retorno

Objeto.

Refresh

Descrição

Atualiza os objetos contidos na MsMGet.

Sintaxe

Refresh()

Hide

Descrição

Esconde a MsMGet.

Sintaxe

Hide()

Show

Descrição

Mostra a MsMGet.

Sintaxe

Show()

Exemplo

```
User Function Exemplo()
Local i
Local oDlg
Local oMsMGet
Local lInit
Local cCampo
DbSelectArea("SA1")
For i := 1 To FCount()
  cCampo := Field(i)
  lInit := .F.
  If ExistIni(cCampo)
    lInit := .t.
    M->(cCampo) := InitPad(SX3->X3_RELACAO)
    If ValType(M->(cCampo)) = "C"
      M->(cCampo) := Padr(M->(cCampo), SX3->X3_TAMANHO)
    EndIf
    If M->(cCampo) == NIL
      lInit := .F.
    EndIf
  EndIf
  If !lInit
    M->(cCampo) := FieldGet(i)
    If ValType(M->(cCampo)) = "C"
      M->(cCampo) := Space(Len(M->(cCampo)))
    ElseIf ValType(M->(cCampo)) = "N"
      M->(cCampo) := 0
    ElseIf ValType(M->(cCampo)) = "D"
      M->(cCampo) := CtoD(" / / ")
    ElseIf ValType(M->(cCampo)) = "L"
      M->(cCampo) := .F.
    EndIf
  EndIf
Next
DEFINE MSDIALOG oDlg TITLE "Exemplo" FROM 00,00 TO 19,80
MSMGET(): NEW("SA1",0,3,,,,,,,,,,,,,T.)
ACTIVATE MSDIALOG oDlg CENTERED ON INIT EnchoiceBar(oDlg,{|| oDlg:End()},{|| oDlg:End()})
```


Return

MBrowse

Monta um Browse com menu de opções.

Sintaxe

```
MBROWSE( [ uPar1 ], [ uPar2 ], [ uPar3 ], [ uPar4 ], cAlias, [ aFixos ], [ cCpo ], [ uPar5 ], [ cFun ], [ nPadrao ], [ aCores ], [ cExplni ], [ cExpFim ], [ nCongela ] ) -> nil
```

Argumento	Obrigat.	Tipo	Descrição
<i>uPar1</i>	Não	N	Parâmetro reservado.
<i>uPar2</i>	Não	N	Parâmetro reservado.
<i>uPar3</i>	Não	N	Parâmetro reservado.
<i>uPar4</i>	Não	N	Parâmetro reservado.
<i>CAlias</i>	Sim	C	Alias do arquivo a ser visualizado no browse.
<i>AFixos</i>	Não	A	Contendo os nomes dos campos fixos pré-definidos pelo programador, obrigando a exibição de uma ou mais colunas.
<i>CCpo</i>	Não	C	Campo a ser validado se está vazio ou não para exibição do bitmap de status.
<i>uPar5</i>	Não	N	Parâmetro reservado.
<i>cFun</i>	Não	C	Função que retornará um valor lógico para exibição do bitmap de status.
<i>nPadrao</i>	Não	N	Número da rotina a executada quando for efetuado um duplo clique em um registros do browse. Caso não seja informado o padrão será executada visualização ou pesquisa.
<i>aCores</i>	Não	A	Este vetor possui duas dimensões, a primeira é a função de validação para exibição do bitmap de status, e a segunda o bitmap a ser exibido.
<i>cExplni</i>	Não	C	Função que retorna o conteúdo inicial do filtro baseada na chave de índice selecionada.
<i>cExpFim</i>	Não	C	Função que retorna o conteúdo final do filtro baseada na chave de índice selecionada.
<i>nCongela</i>	Não	N	Coluna a ser congelado no browse.

Descrição

As variáveis cCadastro e aRotina (*) precisam ser declarados como private acima da chamada da função.

Apenas um dos parâmetros (cCpo, cFun, aColors) deve ser informado.

(*) vetor com as rotinas que serão executadas, nele será definido o tipo de operação a ser executada (inclusão, alteração, exclusão, visualização, pesquisa, ...) sua estrutura é composta de 5 (cinco) dimensões: 1º - Título; 2º - Rotina; 3º - Reservado; 4º - Operação (1 - pesquisa; 2 - visualização; 3 - inclusão; 4 - alteração; 5 - exclusão); 5 - Acesso relacionado a rotina, se esta posição não for informada não validará os acessos.

Exemplo

```
Private cCadastro := 'Cadastro de Clientes'
Private aRotina := { { 'Pesquisar' , 'AxPesqui' , 0, 1 },;
                    { 'Visualizar' , 'AxVisual' , 0, 2 },;
                    { 'Incluir' , 'AxInclui' , 0, 3 },;
                    { 'Alterar' , 'AxAltera' , 0, 4 },;
                    { 'Excluir' , 'AxExcluir' , 0, 5 }}
MBROWSE( , , , 'SA1' , , '!Al_COD' , , , 4 )
```

Funções

AllGroups

Retorna vetor contendo informações dos grupos de usuários.

Sintaxe

ALLGROUPS() -> array

Descrição

A função AllGroups() retorna um vetor principal onde cada elemento refere-se a um grupo de usuários do sistema, estes elementos são compostos de um vetor multidimensional subdividindo as informações dos grupos. Sua estrutura é composta de:

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
2				
	1	Módulo+nível+menu	C	

Exemplo

```
Local aGrupos:= {}
aGrupos:= ALLGROUPS()
```

AllUsers

Retorna vetor contendo informações dos usuários do sistema.

Sintaxe

ALLUSERS() -> array

Descrição

A função AllUsers() retorna um vetor principal onde cada elemento refere-se a um usuário do sistema, estes elementos são compostos de um vetor multidimensional subdividindo as informações dos usuários. Sua estrutura é composta de:

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
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
3				
	1	Módulo+nível+menu	C	

Exemplo

```
Local aUsuario:= {}  
aUsuario:= ALLUSERS()
```

APMsgAlert

Exibe uma mensagem em um diálogo

Sintaxe

APMSGALERT(*cMsg*, [*cTitulo*]) -> nil

Argumento	Obrigat.	Tipo	Descrição
<i>cMsg</i>	Sim	C	Mensagem a ser exibida.
<i>cTitulo</i>	Não	C	Título do diálogo.

Descrição

Esta função exibe uma mensagem em um diálogo para alertar o usuário com um botão para confirmar e um ícone de mensagem de aviso.

Exemplo

```
APMSGALERT("Atenção")
```

APMsgInfo

Exibe uma mensagem em um diálogo

Sintaxe

APMSGINFO(*cMsg*, [*cTitulo*]) -> nil

Argumento	Obrigat.	Tipo	Descrição
<i>cMsg</i>	Sim	C	Mensagem a ser exibida.
<i>cTitulo</i>	Não	C	Título do diálogo.

Descrição

Esta função exibe uma mensagem em um diálogo para alertar o usuário com um botão para confirmar e um ícone de mensagem de aviso.

Exemplo

```
APMSGINFO("Atenção")
```

APMsgNoYes

Exibe uma mensagem em um diálogo

Sintaxe

APMSGNOYES(*cMsg*, [*cTitulo*]) -> lógico

Argumento	Obrigat.	Tipo	Descrição
<i>cMsg</i>	Sim	C	Mensagem a ser exibida.
<i>cTitulo</i>	Não	C	Título do diálogo.

Descrição

Esta função exibe uma mensagem em um diálogo para o usuário com um botão para confirmar e um para cancelar. Caso cancele a função retornará verdadeiro, senão retornará falso.

Exemplo

```
If !APMSGNOYES("Abandonar?")
    Return
EndIf
```

APMsgStop

Exibe uma mensagem em um diálogo

Sintaxe

APMSGSTOP(*cMsg*, [*cTitulo*]) -> nil

Argumento	Obrigat.	Tipo	Descrição
<i>cMsg</i>	Sim	C	Mensagem a ser exibida.
<i>Ctitulo</i>	Não	C	Título do diálogo.

Descrição

Esta função exibe uma mensagem em um diálogo para alertar o usuário com um botão para confirmar e um ícone de mensagem crítica.

Exemplo

```
APMSGSTOP("Atenção")
```

APMsgYesNo

Exibe uma mensagem em um diálogo

Sintaxe

```
APMSGYESNO( cMsg , [ cTitulo ] ) -> lógico
```

Argumento	Obrigat.	Tipo	Descrição
<i>cMsg</i>	Sim	C	Mensagem a ser exibida
<i>cTitulo</i>	Não	C	Título do diálogo.

Descrição

Esta função exibe uma mensagem em um diálogo para o usuário com um botão para confirmar e um para cancelar. Caso confirme a função retornará verdadeiro, senão retornará falso.

Exemplo

```
If APMSGYESNO("Abandonar ?")  
    Return  
EndIf
```

APMsgYesNo

Exibe uma mensagem em um diálogo

Sintaxe

```
APMSGYESNO( cMsg , [ cTitulo ] ) -> lógico
```

Argumento	Obrigat.	Tipo	Descrição
<i>cMsg</i>	Sim	C	Mensagem a ser exibida
<i>cTitulo</i>	Não	C	Título do diálogo.

Descrição

Esta função exibe uma mensagem em um diálogo para o usuário com um botão para confirmar e um para cancelar. Caso confirme a função retornará verdadeiro, senão retornará falso.

Exemplo

```

If APMSGYESNO("Abandonar ?")
    Return
EndIf

```

Cabec

Imprime cabeçalho personalizado nos relatórios.

Sintaxe

CABEC(*cTitulo*, *cCabec1*, *cCabec2*, *cPrograma*, *cTamanho*, [*nFormato*], [*uPar*], [*lPerg*]) -> nil

Argumento	Obrigat.	Tipo	Descrição
<i>cTitulo</i>	Sim	C	Título do relatório.
<i>cCabec1</i>	Sim	C	Primeira linha do cabeçalho.
<i>cCabec2</i>	Sim	C	Segunda linha do cabeçalho.
<i>cPrograma</i>	Sim	C	Nome do relatório.
<i>cTamanho</i>	Sim	C	Tamanho da página P,M,G.
<i>nFormato</i>	Não	N	Para imprimir comprimido informe 15
<i>uPar</i>	Não	U	Reservado
<i>lPerg</i>	Não	L	Se verdadeiro, (.T.) imprime as perguntas no início do relatório.

Descrição

A função Cabec() imprime o cabeçalho personalizado de acordo com o conteúdo dos parâmetros cCabec1 e cCabec2. O logo impresso no cabeçalho é uma imagem de extensão ".bmp", cujo nome está associado com o empresa corrente. Ex: 'LGRL01.BMP', '01' é a empresa.

Exemplo

```

User Function <nome-da-função>( )
Local cProgram := 'Exemplo' // nome do relatório
Local cAlias := 'XXX' // alias do arquivo
Local cPerg := 'XXXXXX' // grupo de perguntas
Local cTitulo := 'Titulo do relatório'
Local cDesc1 := 'Descrição'
Local cDesc2 := 'continuação da descrição'
Local cDesc3 := 'continuação da descrição'
Local lDic := .F. // não utiliza dicionário
Local aOrd := { '1º Ordem', '2º ordem', '3º ordem' }

```



```

Local lCompres :=.F.
Local cTam := "G"
Private m_pag := 1
Private aReturn := { "Zebrado", 1,"Administração", 1, 2, 1, "",1 }
Private wrel
Pergunte( cPerg, .F. )

wrel := SetPrint( cAlias, cProgram, cPerg, @cTitulo, cDesc1, cDesc2, cDesc3, lDic, aOrd ,
lCompres, cTam )
If nLastKey == 27
    Set Filter To
    Return
EndIf
SetDefault( aReturn, cAlias )
If nLastKey == 27
    Set Filter To
    Return
EndIf
RptStatus( { |lFim| Imprime( @lFim, cAlias, cTitulo, cProgram, cTamanho ) }, cTitulo )
Return

// rotina de impressão
Static Function Imprime( lFim, cAlias, cTitulo, cProgram, cTamanho )
Local nLin := 80
Local cCabec1 := 'Cabecalho 1'
Local cCabec2 := 'Cabecalho 2'
dbSelectArea( cAlias )
SetRegua( RecCount() )
While !Eof() .And. ( XX_COD >= MV_PAR01 .And. XX_COD <= MV_PAR02 )
    If lFim
        @Prow()+1,001 PSAY "CANCELADO PELO OPERADOR"
        Exit
    EndIf
    If nLin > 58
        CABEC( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
        nLin := 8
    EndIf
    @ nLin,000 PSAY __PrtFatLine()
    @ nLin++,001 PSAY ...
    @ nLin++,001 PSAY ...
    @ nLin++,001 PSAY ...
    @ nLin,000 PSAY __PrtThinLine()
    IncRegua()
    dbSkip()
End
IF nLin != 80
    nLin++
    If nLin > 60
        CABEC( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
    Endif

```

```
@ nLin,000 PSAY __PrtThinLine()  
Roda(,, cTamanho ) // Rodapé  
EndIf  
If aReturn[5] = 1  
    Set Printer TO  
    dbCommitAll()  
    OurSpool( wrel )  
Endif  
MS\_FLUSH()  
Return
```

Capital

Corrige maiúsculas e minúsculas de uma frase.

Sintaxe

CAPITAL(*cTexto*) -> caracter

Argumento	Obrigat.	Tipo	Descrição
cTexto	Sim	C	Texto a ser corrigido.

Descrição

Esta função coloca a primeira letra de cada palavra em maiúscula e o restante em minúscula em toda a frase.

Exemplo

[APMsgInfo](#)(CAPITAL("FrAsE InCoRrEtA"))

CloseBrowse

Fecha a MBrowse ou a MarkBrow.

Sintaxe

CLOSEBROWSE() -> lógico

Descrição

Esta função fecha a MBrowse ou MarkBrow dependendo de qual estiver ativa.

Conpad1

Exibe a tela de consulta padrão.

Sintaxe

CONPAD1([*uPar1*], [*uPar2*], [*uPar3*], *cAlias*, [*cCampoRet*], [*uPar4*], [*IVisual*]) -> lógico

Argumento	Obrigat.	Tipo	Descrição
uPar	Não	U	Parâmetro reservado.
uPar2	Não	U	Parâmetro reservado.
uPar3	Não	U	Parâmetro reservado.
cAlias	Sim	C	Consulta padrão cadastrada no Dicionário de Dados (SXB) a ser utilizada.
cCampoRet	Não	C	Nome da variável ou campo que receberá o retorno da consulta padrão.
uPar4	Não	U	Parâmetro Reservado.
IVisual	Não	L	Indica se será somente para visualização.

Descrição

Esta função exibe a tela de consulta padrão baseada no Dicionário de Dados (SXB).

Exemplo

CONPAD1(, , , "SA1" , , , .F.)

Enchoicebar

Cria barra de botões padrão na janela

Sintaxe

ENCHOICEBAR(*oDlg*, *bOk*, *bCancelar*, [*IMensApag*] , [*aBotoes*]) -> nil

Argumento	Obrigat.	Tipo	Descrição
-----------	----------	------	-----------

<i>oDlg</i>	Sim	O	Janela onde a barra será criada.
<i>bOk</i>	Sim	B	Bloco executado quando clicado botão Ok.
<i>bCancelar</i>	Sim	B	Bloco executado quando clicado.
<i>lMensApag</i>	Não	L	Indica se ao clicar no botão Ok aparecerá uma tela de confirmação de exclusão. Valor padrão falso
<i>aBotoes</i>	Não	A	Vetor com informações para criação de botões adicionais na barra no formato {bitmap, bloco de código, mensagem}.

Descrição

Esta função cria uma barra com botões padrão e outros passados como parâmetro na janela também passada por parâmetro. A *EnchoiceBar* deve ser chamada antes do *ACTIVATE* da janela.

Exemplo

```
User Function <nome-do-programa>()
Local oDlg
DEFINE MSDIALOG oDlg TITLE "Exemplo" FROM 00,00 TO 300,400 PIXEL
ACTIVATE MSDIALOG oDlg CENTERED ON INIT,;
ENCHOI CEBAR(oDlg,{|| APMsgInfo("Ok")}, {|| APMsgInfo("Cancelar")}, , {"BMPINCLUIR",{||
APMsgInfo("Teste")},"Teste"}})
Return
```

FileNoExt

Retorna o nome de um arquivo sem a extensão.

Sintaxe

FILENOEXT(*cTexto*) -> caracter

Argumento	Obrigat.	Tipo	Descrição
<i>cTexto</i>	Sim	C	Texto contendo o nome do arquivo

Descrição

A função *FileNoExt()* retorna o nome de um arquivo contido em uma string, ignorando a extensão.

Exemplo

```
Local cString := '\SIGAADV\ARQZZZ.DBF'
```

```
cString := FileNoExt( cString )  
// retorno '\SIGAADV\ARQZZZ'
```

Final

Utilizada para finalizar o sistema.

Sintaxe

FINAL([cMensagem1], [cMensagem2]) -> lógico

Argumento	Obrigat.	Tipo	Descrição
cMensagem1	Não	C	Primeira mensagem.
cMensagem2	Não	C	Segunda mensagem.

Descrição

Esta função executa 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.

Exemplo

```
User Function <nome-do-programa>( cUsuario, cSenha )  
Local cMensag1 := "Usuário inválido!"  
Local cMensag2 := "Opção disponível para usuários Administradores!"  
If !PswAdmin( cUsuario, cSenha )  
    FINAL( cMensag1, cMensag2 )  
EndIf  
Return
```

FTPConnect

Cria conexão com servidor FTP.

Sintaxe

FTPCONNECT(cServidor, nPorta, cUsuario, cSenha) -> lógico

Descrição

A função FTPConnect() retorna verdadeiro se a operação for realizada com sucesso. Se existir uma conexão ativa e for solicitada uma nova, retornará falso.

Argumento	Obrigat.	Tipo	Descrição
cServidor	Sim	C	Endereço do servidor
nPorta	Não	N	Número da porta utilizada para conectar no servidor, por default utiliza a porta padrão de FTP.
cUsuario	Não	C	Nome do usuário utilizado para conectar no servidor. Por default utiliza o usuário "Anonymous".
cSenha	Não	C	Senha do usuários utilizado para conectar no servidor. Por default utiliza o usuário "Anonymous".

Exemplo

```
Local cServidor
Local cCurDir
cServidor := 'ftp.caminhodoservidor.com.br'
If !FTPConnect( cServidor )
    APMsgInfo( 'Falha na conexão!' )
EndIf
```

FTPDDirChange

Altera o diretório corrente do FTP.

Sintaxe

FTPDIRCHANGE(cDiretorio) -> lógico

Argumento	Obrigat.	Tipo	Descrição
cDiretorio	Sim	C	Nome do diretório.

Descrição

A função FTPDirChange () retornará verdadeiro (.T.) se a operação for realizada com sucesso.

Exemplo

```
Local cServidor
Local aArqs
Local aDirs
cServidor := 'ftp.caminhodoservidor.com.br'
FTPDisconnect()
If !FTPConnect( cServidor )
    APMsgInfo( 'Falha na conexão!' )
EndIf
If FTPDirChange('diretorio')
    aDirs := FTPDirectory( '.*.*', 'D' )
    aArqs := FTPDirectory( '.*.*' )
EndIf
```

EndIf

FTPDirectory

Cria um vetor com informações de diretórios e arquivos do FTP.

Sintaxe

FTPDIRECTORY(*cMascara*, [*cAtributo*]) -> array

Argumento	Obrigat.	Tipo	Descrição
<i>cMascara</i>	Sim	C	Máscara dos arquivos a serem pesquisados.
<i>cAtributo</i>	Não	C	Se for informado "D" a função retornará somente diretórios, se não for informado retornará somente arquivos.

Descrição

A função FTPDirectory() retorna um vetor contendo informações dos diretórios e arquivos contidos no FTP.

Exemplo

```
Local cServidor
Local aArqs
Local aDirs
cServidor := 'ftp.caminhodoserivor.com.br'
FTPDisconnect()
If !FTPConnect( cServidor )
    APMsgInfo( 'Falha na conexão!' )
EndIf
If FTPDirChange('diretorio')
    aDirs := FTPDI RECTORY( '*.*', 'D' )
    aArqs := FTPDI RECTORY( '*.*' )
EndIf
```

FTPDisconnect

Elimina conexão atual.

Sintaxe

FTPDISCONNECT() -> lógico

Descrição

A função FTPDisconnect() retornará verdadeiro (.T.) se a operação for realizada com sucesso. Se não existir uma conexão a função retornará falso (.F.). Se a operação for realizada com sucesso a função retorna verdadeiro (.T.).

Exemplo

```
Local cServidor
Local aArqs
Local aDirs
cServidor := 'ftp.caminhodoservidor.com.br'
FTPDISCONNECT()
If !FTPConnect( cServidor )
    APMsgInfo( 'Falha na conexão!' )
EndIf
FTPDirChange('diretorio')
aDirs := FTPDirectory( '.*', 'D' )
aArqs := FTPDirectory( '.*' )
```

FTPDownload

Copia um arquivo no servidor FTP para o servidor local.

Sintaxe

FTPDOWNLOAD(cArqDest, cArqOrig) -> lógico

Argumento	Obrigat.	Tipo	Descrição
cArqDest	Sim	C	Caminho e nome do arquivo a ser gravado na máquina.
CArqOrig	Sim	C	Arquivo no servidor a ser copiado.

Descrição

A função FTPDownload() copia um arquivo no servidor FTP para uma máquina local em um diretório (informado no parâmetro cArqDest) abaixo do RootPath do Protheus.

Exemplo

```
Local cServidor
cServidor := 'ftp.caminhodoservidor.com.br'
FTPDISCONNECT()
If !FTPConnect( cServidor )
    APMsgInfo( 'Falha na conexão!' )
EndIf
If FTPDirChange('diretorio')
    If !FTPDOWNLOAD( '\DIRETORIO\ARQ00001.ARQ', 'ARQ00001.ARQ' )
        APMsgInfo( 'Problemas ao copiar arquivo!' )
    EndIf
EndIf
```

FTPErase

Apaga arquivo no servidor FTP.

Sintaxe

FTP~~E~~RASE(*cArquivo*) -> lógico

Argumento	Obrigat.	Tipo	Descrição
cArquivo	Sim	C	Nome do arquivo

Descrição

A função FTP~~E~~rase() apaga arquivo no diretório corrente do FTP. Se a operação for realizada com sucesso a função retornará verdadeiro (.T.).

Exemplo

```
Local cServidor
Local aArgs
Local aDirs
cServidor := 'ftp.caminhodoserivor.com.br'
FTPDisconnect()
If !FTPConnect( cServidor )
    APMsgInfo( 'Falha na conexão!' )
EndIf
If FTPDirChange('diretorio')
    If !FTPERASE( 'Arquivo.ext' )
        APMsgInfo('Problemas ao apagar o arquivo!' )
    EndIf
EndIf
```

FTPGetCurDir

Retorna o diretório corrente no FTP.

Sintaxe

FTPGETCURDIR() -> caracter

Exemplo

```
Local cServidor
Local cCurDir
cServidor := 'ftp.caminhodoserivor.com.br'
FTPDisconnect()
If !FTPConnect( cServidor )
    APMsgInfo( 'Falha na conexão!' )
EndIf
cCurDir := FTPGETCURDIR()
```

FTPRenameFile

Renomeia arquivo no servidor FTP.

Sintaxe

FTPRENAMEFILE(*cArqAtual*, *cArqNovo*) -> lógico

Argumento	Obrigat.	Tipo	Descrição
<i>cArqAtual</i>	Sim	C	Nome do arquivo a ser renomeado.
<i>cArqDest</i>	Sim	C	Novo nome do arquivo.

Descrição

A função FTPRenameFile() renomeia um arquivo no diretório corrente do servidor FTP. Se a operação for realizada com sucesso a função retornará verdadeiro (.T.).

Exemplo

```
Local cServidor
cServidor := 'ftp.caminhodoserivor.com.br'
FTPDisconnect()
If !FTPConnect( cServidor )
    APMsgInfo( 'Falha na conexão!' )
EndIf
If FTPDirChange('diretorio')
    If !FTPRENAMEFILE( 'ARQ00001.ARQ', 'ARQ00002.ARQ' )
        APMsgInfo( 'Problemas ao renomear arquivo!' )
    EndIf
EndIf
```

FTPUpload

Copia um arquivo na máquina local para o servidor FTP.

Sintaxe

FTPUPLOAD(*cArqOrig*, *cArqDest*) -> lógico

Argumento	Obrigat.	Tipo	Descrição
<i>cArqOri</i>	Sim	C	Caminho e nome do arquivo na máquina a ser copiado para o servidor FTP.
<i>cArqDest</i>	Sim	C	Nome do arquivo a ser gravado no servidor FTP.

Descrição

A função FTPUpload() copia um arquivo na máquina local para o diretório corrente no servidor FTP. O arquivo a ser copiado deve estar abaixo do RootPath do Protheus. Se a operação for realizada com sucesso a função retornará verdadeiro (.T.).

Exemplo

```
Local cServidor
cServidor := 'ftp.caminhodoserivor.com.br'
FTPDisconnect()
If !FTPConnect( cServidor )
    APMsgInfo( 'Falha na conexão!' )
EndIf
If FTPDirChange('diretorio')
    If !FTPUPLOAD( '\DIRETORIO\ARQ00001.ARQ', 'ARQ00001.ARQ' )
        APMsgInfo( 'Problemas ao copiar arquivo!' )
    EndIf
EndIf
```

FunDesc

Retorna descrição de uma opção do menu.

Sintaxe

FUNDESC() -> caracter

Descrição

A função FunName() retornará a descrição de uma opção selecionada no menu do Siga.

Exemplo

```
Local cDescr
cDescr := FUNDESC()
```

FunName

Retorna nome de uma função do menu.

Sintaxe

FUNNAME() -> caracter

Descrição

A função FunName() retornará o nome de uma função executada a partir de um menu do Siga

Exemplo

```
Local cFunction
cFunction := FUNNAME()
```

GetCountryList

Retorna vetor contendo informações dos países localizados.

Sintaxe

GETCOUNTRYLIST() -> array

Descrição

O vetor retornado possui três dimensões, a primeira refere-se a sigla dos países, o segundo ao nome do país e o terceiro 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
```

GetMark

Retorna string de caracteres aleatórios.

Sintaxe

GETMARK([IMaiusc]) -> caracter

Argumento	Obrigat.	Tipo	Descrição
IMaiusc	Não	L	Se verdadeiro (.T.) retorna somente caracteres em maiúsculos.

Descrição

A GetMark() é utilizada junto a função MarkBrow() onde são utilizados combinações de caracteres para controle de marcas.

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

```

GetMv

Retorna o conteúdo de um parâmetro cadastrado no SX6.

Sintaxe

GETMV(cPar01, [IPar02], [uPar03]) -> variável

Argumento	Obrigat.	Tipo	Descrição
cPar1	Sim	C	Nome do parâmetro a ser pesquisado
IPar02	Não	L	Define se a GetMv deve retornar o conteúdo do parâmetro, .F. (Falso) ou apenas verificar se o parâmetro existe, .T. (Verdadeiro). Valor default .F. (Falso).
uPar03	Não	U	Valor default que deve ser retornado pela GetMv quando os parâmetro solicitado não existir. O valor desse parâmetro pode ser caracter, numérico, lógico ou data.

Descrição

O retorno da função depende do tipo informado na cadastro do parâmetro e da configuração dos argumentos IPar02 e uPar03.

Se IPar02 for passado como .T. (Verdadeiro), o retorno da função será um valor lógico indicando se o parâmetro existe.

Quando uPar03 for informado, caso o parâmetro informado em cPar01 não exista, o retorno da função será o valor informado em uPar03 caso contrário retorna o conteúdo do parâmetro. Se uPar03 for informado o conteúdo de IPar02 será desconsiderado.

Exemplo

```

Local cValor := ""
// Retorna o conteúdo do parâmetro
cValor := GETMV( "MV_ESTADO" )
// Verifica se o parâmetro existe
If ( GETMV( "MV_ESTADO", .T. ) )
    ApMsgInfo( "O Parâmetro MV_ESTADO existe !" )
Else
    ApMsgStop( "O Parâmetro MV_ESTADO não existe !" )
EndIf
// Retorna o conteúdo do parâmetro
// se não encontrar retorna o valor default passado
cValor := GETMV( "MV_ESTADO", , "SP" )

```

IncProc

Incrementa régua de progressão.

Sintaxe

INCPROC() -> nil

Descrição

Para incrementar a régua criada pela função Processa(), utilizamos a função IncProc()

Exemplo

```
User Function <nome-da-função>( )
Local bAcao := { |lFim| Exemplo(@lFim) }
Local cTitulo := ''
Local cMsg := 'Processando'
Local lAborta := .T.
Processa( bAcao, cTitulo, cMsg, lAborta )
Return

Static Function Exemplo(lFim)
Local nI
ProcRegua(10000)
For nI := 1 To 10000
    If lFim
        Exit
    EndIf
    INCPROC()
Next nI
Return
```

IncRegua

Incrementa valores na régua de progressão criada pela função RptStatus().

Sintaxe

INCREGUA() -> nil

Descrição

Após executar as funções RptStatus() e SetRegua(), para incrementar valores na régua utilizamos a função IncRegua().

Exemplo

```

User Function <nome-da-função>( )
Local cProgram := 'Exemplo' // nome do relatório
Local cAlias := 'XXX' // alias do arquivo
Local cPerg := 'XXXXXX' // grupo de perguntas
Local cTitulo := 'Titulo do relatório'
Local cDesc1 := 'Descrição'
Local cDesc2 := 'continuação da descrição'
Local cDesc3 := 'continuação da descrição'
Local lDic := .F. // não utiliza dicionário
Local aOrd := { '1º Ordem', '2º ordem', '3º ordem' }
Local lCompres := .F.
Local cTam := "G"
Private m_pag := 1
Private aReturn := { "Zebrado", 1, "Administração", 1, 2, 1, "", 1 }
Private wrel

Pergunte( cPerg, .F. )

wrel := SetPrint( cAlias, cProgram, cPerg, @cTitulo, cDesc1, cDesc2, cDesc3, lDic, aOrd ,
lCompres, cTam )
If nLastKey == 27
    Set Filter To
    Return
EndIf
SetDefault( aReturn, cAlias )
If nLastKey == 27
    Set Filter To
    Return
EndIf
RptStatus( { |lFim| Imprime( @lFim, cAlias, cTitulo, cProgram, cTamanho ) }, cTitulo )
Return

// rotina de impressão
Static Function Imprime( lFim, cAlias, cTitulo, cProgram, cTamanho )
Local nLin := 80
Local cCabec1 := 'Cabecalho 1'
Local cCabec2 := 'Cabecalho 2'
dbSelectArea( cAlias )
SetRegua( RecCount() )
While !Eof() .And. ( XX_COD >= MV_PAR01 .And. XX_COD <= MV_PAR02 )
    If lFim
        @Prow()+1,001 PSAY "CANCELADO PELO OPERADOR"
        Exit
    EndIf
    If nLin > 58
        Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
        nLin := 8
    EndIf
    @ nLin,000 PSAY __PrtFatLine()
    @ nLin++,001 PSAY ...
    @ nLin++,001 PSAY ...
    @ nLin++,001 PSAY ...
    @ nLin,000 PSAY __PrtThinLine()
    ! NCREGUA()
    dbSkip()
End
IF nLin != 80
    nLin++
    If nLin > 60
        Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
    Endif
    @ nLin,000 PSAY __PrtThinLine()
    Roda( 0, cTitulo, cTamanho ) // Rodapé
EndIf
If aReturn[5] = 1
    Set Printer TO
    dbCommitAll()
    OurSpool( wrel )
Endif
MS\_FLUSH()

```

Return

IndRegua

Cria índice temporário com expressão de filtro.

Sintaxe

INDREGUA(*cAlias*, *cIndice*, *cExpress*, [*xOrdem*] , [*cFor*], [*cMens*], [*IExibir*]) -> nil

Argumento	Obrigat.	Tipo	Descrição
cAlias	Não	C	Alias da tabela que o índice será criado.
cIndece	Sim	C	Nome do arquivo para criação do índice.
cExpress	Sim	C	Expressão do índice.
xOrdem	Não	C	Indica se a ordem será crescente ou decrescente.
IExibir	Não	L	Indica se exibirá o diálogo de progressão.
cFor	Não	C	Expressão de filtro.
cMens	Não	C	Mensagem do diálogo de progressão.

Descrição

Esta função cria um índice temporário para o alias especificado podendo ou não ter um filtro e também podendo ser decrescente se o parâmetro xOrdem for especificado como "D". Durante a criação do índice um diálogo com uma barra de progressão será criada podendo esta ser omitida.

Exemplo

```
User Function Exemplo()
Local cArquivo
Local cChave
Local cFor
Local nIndex
DbSelectArea("SA1")
cArquivo := CriaTrab(, .F.)
cChave := "A1_NOME"
cFor := "!Empty(A1_NOME)"
INDREGUA("SA1", cArquivo, cChave, , cFor)
DbSelectArea("SA1")
nIndex := RetIndex("SA1")
#IFDEF TOP
    DbSetIndex(cArquivo+OrdBagExt())
#ENDIF
DbSetOrder(nIndex+1)
.
.
.
DbSelectArea("SA1")
RetIndex("SA1")
```



```
FErase(cArquivo+OrdBagExt())
Return
```

MarkBRefresh

Atualiza a MarkBrow.

Sintaxe

MARKBREFRESH() -> nil

Descrição

Esta função atualiza o browse da MarkBrow.

MarkBrow

Monta um Browse onde as linhas podem ser marcadas ou desmarcadas.

Sintaxe

```
MARKBROW( cAlias, cCampo, [ cCpo ], [ aCampos ], [ lInverte ], cMarca, [ cCtrlM ], [ uPar ], [ cExpIni ], [ cExpFim ], [ cAval ] ) -> nil
```

Argumento	Obrigat.	Tipo	Descrição
<i>cAlias</i>	Sim	C	Alias do arquivo a ser exibido no browse.
<i>cCampo</i>	Sim	C	Campo do arquivo onde será feito o controle (gravação) da marca.
<i>cCpo</i>	Não	C	Campo onde será feita a validação para marcação e exibição do bitmap de status.
<i>aCampos</i>	Não	A	Colunas a serem exibidas.
<i>lInvert</i>	Não	L	Inverte a marcação.
<i>cMarca</i>	Sim	C	String a ser gravada no campo especificado para marcação.
<i>cCtrlM</i>	Não	C	Função a ser executada caso deseje marcar todos elementos.
<i>uPar</i>	Não	L	Parâmetro reservado.
<i>cExpIni</i>	Não	C	Função que retorna o conteúdo inicial do filtro baseada na chave de índice selecionada.
<i>cExpFim</i>	Não	C	Função que retorna o conteúdo final do filtro baseada na chave de índice selecionada.

<i>cAval</i>	Não	C	Função a ser executada no duplo clique em um elemento no browse.
--------------	-----	---	--

Descrição

A função MarkBrow() permite que os elementos de um browse sejam marcados ou desmarcados.

As variáveis cCadastro e aRotina (*) precisam ser declarados como private acima da chamada da função.

O vetor informado no parâmetro aCampos deve conter as seguintes dimensões: 1º – nome do campo; 2º - Nada (Nil); 3º - Título do campo; 4º - Máscara (picture).

(*) vetor com as rotinas que serão executadas, nele será definido o tipo de operação a ser executada (inclusão, alteração, exclusão, visualização, pesquisa, ...) sua estrutura é composta de 5 (cinco) dimensões: 1º - Título; 2º - Rotina; 3º – Reservado; 4º – Operação (1 - pesquisa; 2 - visualização; 3 - inclusão; 4 - alteração; 5 - exclusão); 5 – Acesso relacionado a rotina, se esta posição não for informada nenhum acesso será validado.

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

// Grava marca no campo
Function Mark()
If IsMark( 'CB_OK', cMarca )
    RecLock( 'SCB', .F. )
    Replace CB_OK With Space(2)
    MsUnLock\(\)
Else
    RecLock( 'SCB', .F. )
    Replace CB_OK With cMarca
    MsUnLock()
EndIf
Return

// Grava marca em todos os registros validos
Function MarkAll()
Local nRecno := Recno()
dbSelectArea( 'SCB' )
dbGotop()
While !Eof()
    Mark()
    dbSkip()
End
dbGoto( nRecno )
Return
```

[Ms_Flush](#)

Descarrega spool de impressão.

Sintaxe

MS_FLUSH() -> nil

Descrição

Após os comandos de impressão as informações ficam armazenadas no spool e são descarrega em seus destinos através da função Ms_Flush().

Exemplo

```
User Function <nome-da-função>( )
Local cProgram := 'Exemplo' // nome do relatório
Local cAlias := 'XXX' // alias do arquivo
Local cPerg := 'XXXXXX' // grupo de perguntas
Local cTitulo := 'Titulo do relatório'
Local cDesc1 := 'Descrição'
Local cDesc2 := 'continuação da descrição'
Local cDesc3 := 'continuação da descrição'
Local lDic := .F. // não utiliza dicionário
Local aOrd := { '1º Ordem', '2º ordem', '3º ordem' }
Local lCompres := .F.
Local cTam := "G"
Private m_pag := 1
Private aReturn := { "Zebrado", 1, "Administracao", 1, 2, 1, "", 1 }
Private wrel

Pergunte( cPerg, .F. )
wrel := SetPrint( cAlias, cProgram, cPerg, @cTitulo, cDesc1, cDesc2, cDesc3, lDic, aOrd ,
lCompres, cTam )
If nLastKey == 27
    Set Filter To
    Return
EndIf
SetDefault( aReturn, cAlias )
If nLastKey == 27
    Set Filter To
    Return
EndIf
RptStatus( { |lFim| Imprime( @lFim, cAlias, cTitulo, cProgram, cTamanho ) }, cTitulo )
Return

// rotina de impressão
Static Function Imprime( lFim, cAlias, cTitulo, cProgram, cTamanho )
Local nLin := 80
Local cCabec1 := 'Cabecalho 1'
Local cCabec2 := 'Cabecalho 2'
dbSelectArea( cAlias )
SetRegua( RecCount() )
While !Eof() .And. ( XX_COD >= MV_PAR01 .And. XX_COD <= MV_PAR02 )
    If lFim
        @Prow()+1,001 PSAY "CANCELADO PELO OPERADOR"
        Exit
    EndIf
    If nLin > 58
        Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
        nLin := 8
    EndIf
    @ nLin,000 PSAY __PrtFatLine()
    @ nLin++,001 PSAY ...
    @ nLin++,001 PSAY ...
    @ nLin++,001 PSAY ...
    @ nLin,000 PSAY __PrtThinLine()
    IncRegua()
    dbSkip()
End
IF nLin != 80
```

```

nLin++
If nLin > 60
    Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
Endif
@ nLin,000 PSAY __PrtThinLine()
Roda( 0, cTitulo, cTamanho ) // Rodapé
Endif
If aReturn[5] = 1
    Set Printer TO
    dbCommitAll()
    OurSpool( wrel )
Endif
MS_FLUSH()
Return

```

MsAppend

Adiciona registros de um arquivo para outro.

Sintaxe

MSAPPEND([cArqDest], cArqOrig) -> lógico

Argumento	Obrigat.	Tipo	Descrição
cArqDest	Não	C	Se o RDD corrente for DBFCDX os registros serão adicionados na área selecionada, caso contrário o arquivo destino terá que ser informado.
cArqOrig	Não	C	Nome do arquivo origem contendo os registros a serem adicionados.

Descrição

A função MsAppend() adiciona registros de um arquivo para outro, respeitando a estrutura das tabelas. Se a operação for realizada com sucesso o função retornará verdadeiro (.T.).

Exemplo

```

dbSelectArea( 'XXX' )
MSAPPEND( , 'ARQ00001' )

```

MsCopyFile

Executa copia binária de um arquivo.

Sintaxe

MSCOPYFILE(cArqOrig, cArqDest) -> lógico

Argumento	Obrigat.	Tipo	Descrição
<i>cArqOrig</i>	Sim	C	Nome do arquivo origem e a extensão.
<i>cArqDest</i>	Sim	C	Nome do arquivo destino e a extensão.

Descrição

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

Realiza copia de um arquivo de dados.

Sintaxe

MSCOPYTO([*cArqOrig*], *cArqDest*) -> lógico

Argumento	Obrigat.	Tipo	Descrição
<i>cArqOrig</i>	Não	C	Nome do arquivo origem e a extensão se o ambiente for Top o parâmetro passará a ser obrigatório.
<i>cArqDest</i>	Sim	C	Nome do arquivo destino e a extensão.

Descrição

A função MsCopyTo() copia os registros de uma base de dados para outra, criando o arquivo destino de acordo com a estrutura da base de dados origem.

Se a operação for realizada com sucesso o 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

Cria arquivo de dados.

Sintaxe

MSCREATE(*cArquivo*, *aEstrut*, [*cDriver*]) -> lógico

Argumento	Obrigat.	Tipo	Descrição
<i>cArquivo</i>	Não	C	Nome do arquivo.
<i>aEstrut</i>	Sim	A	Estrutura do arquivo.
<i>cDriver</i>	Não	C	RDD do arquivo.

Descrição

A função MsCreate() cria um arquivo (tabela) de acordo com a estrutura informada no parâmetro aStrut (*). 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.

Se o arquivo for criado com sucesso a função retornará verdadeiro (.T.).

(*) vetor contendo a estrutura da tabela: 1º - caracter, nome do campo; 2º - caracter, tipo do campo; 3º - numérico, tamanho do campo; 4º - numérico, decimais.

Exemplo

```
Local cTarget := '\sigaadv\'
Local aStrut
aStrut := { { 'Campo', 'C', 40, 0 } }
If MSCREATE( cTarget+'ARQ1001', aStrut )
    APMsgInfo('Criado com sucesso!')
Else
    APMsgInfo('Problemas ao criar o arquivo!')
EndIf
```

MsErase

Deleta arquivo.

Sintaxe

MSERASE(*cArquivo*, [*cIndice*], [*cDriver*]) -> lógico

Argumento	Obrigat.	Tipo	Descrição
-----------	----------	------	-----------

<i>cArquivo</i>	Sim	C	Nome do arquivo e a extensão.
<i>cIndice</i>	Não	C	Nome do arquivo de índice e a extensão.
<i>cDriver</i>	Não	C	RDD do arquivo, se não for informado assumirá o RDD corrente como padrão.

Descrição

A função `MsErase()` retornará verdadeiro (.T.) se a operação for realizada com sucesso.

Exemplo

```
Local cArquivo := 'SX2ZZZ.DBF'
Local cIndice  := 'SX2ZZZ'+ OrdBagExt()
If MSERASE( cArquivo, cIndice )
    APMsgInfo( 'Arquivo deletado com sucesso!' )
Else
    APMsgInfo( 'Problemas ao deletar arquivo!' )
EndIf
```

MsFile

Verifica existência de um arquivo.

Sintaxe

MSFILE(*cArquivo*, [*cIndice*], [*cDriver*]) -> lógico

Argumento	Obrigat.	Tipo	Descrição
<i>cArquivo</i>	<i>Sim</i>	C	Nome do arquivo origem e a extensão.
<i>cIndice</i>	Não	C	Nome do arquivo de índice e a extensão.
<i>cDriver</i>	Não	C	RDD do arquivo, se não for informado assumirá o RDD corrente como padrão.

Descrição

A função `MsFile()` retornará verdadeiro (.T.) se a operação for realizada com sucesso.

Exemplo

```
Local cArquivo := 'SX2ZZZ.DBF'
Local cIndice  := 'SX2ZZZ'+ OrdBagExt()
If !MSFILE( cArquivo, cIndice )
```

```
        APMsgInfo( 'Arquivo não encontrado!' )  
    EndIf
```

MsRename

Renomeia arquivo de acordo com RDD corrente.

Sintaxe

MSRENAME(*cArqOrig*, *cArqDest*) -> lógico

Argumento	Obrigat.	Tipo	Descrição
cArqOrig	Sim	C	Nome do arquivo origem e a extensão.
cArqDest	Sim	C	Nome do arquivo destino e a extensão.

Descrição

A função MsRename() retornará verdadeiro (.T.) se efetuar a operação com sucesso.

Exemplo

```
Local cTarget := '\sigaadv\  
Local cArqOrig := 'ARQ00001.DBF'  
Local cArqDest := 'ARQ00002.DBF'  
If MSRENAME( cTarget + cArqOrig, cTarget + cArqDest )  
    APMsgInfo('Arquivo renomeado com sucesso!')  
Else  
    APMsgInfo('Problemas ao renomear o arquivo ' + cArqOrig + '!')  
EndIf
```

MsUnlock

Libera lock de registro.

Sintaxe

MSUNLOCK() -> nil

Descrição

A função MsUnlock() libera os registros bloqueados pela função RecLock().

Não retorna valores.

Exemplo


```
RecLock( 'XXX' ,.F. )
Replace Campo With '000001'
MSUNLOCK()
```

OurSpool

Gerenciador de impressão.

Sintaxe

```
OURSPOOL( [ cArquivo ] ) -> nil
```

Argumento	Obrigat.	Tipo	Descrição
cArquivo	Sim	C	Relatório gerado em disco

Descrição

A função OurSpool() executa o gerenciador de impressão do Siga, carregando os relatórios gerados no diretório configurado através parâmetro MV_RELT no dicionário SX6. Caso quiser visualizar um relatório específico, informe o nome no parâmetro cArquivo.

Exemplo

```
User Function <nome-da-função>( )
Local cProgram := 'Exemplo' // nome do relatório
Local cAlias := 'XXX' // alias do arquivo
Local cPerg := 'XXXXXX' // grupo de perguntas
Local cTitulo := 'Titulo do relatório'
Local cDesc1 := 'Descrição'
Local cDesc2 := 'continuação da descrição'
Local cDesc3 := 'continuação da descrição'
Local lDic := .F. // não utiliza dicionário
Local aOrd := { '1º Ordem', '2º ordem', '3º ordem' }
Local lCompres := .F.
Local cTam := "G"
Private m_pag := 1
Private aReturn := { "Zebrado", 1,"Administracao", 1, 2, 1, "",1 }
Private wrel

Pergunte( cPerg, .F. )
wrel := SetPrint( cAlias, cProgram, cPerg, @cTitulo, cDesc1, cDesc2, cDesc3, lDic, aOrd ,
lCompres, cTam )
If nLastKey == 27
    Set Filter To
    Return
EndIf
SetDefault( aReturn, cAlias )
If nLastKey == 27
    Set Filter To
    Return
EndIf
RptStatus( {|lFim| Imprime( @lFim, cAlias, cTitulo, cProgram, cTamanho ) }, cTitulo )
Return

// rotina de impressão
Static Function Imprime( lFim, cAlias, cTitulo, cProgram, cTamanho )
Local nLin := 80
Local cCabec1 := 'Cabecalho 1'
```

```

Local cCabec2 := 'Cabecalho 2'
dbSelectArea( cAlias )
SetRegua( RecCount() )
While !Eof() .And. ( XX_COD >= MV_PAR01 .And. XX_COD <= MV_PAR02 )
  If lFim
    @Prow()+1,001 PSAY "CANCELADO PELO OPERADOR"
    Exit
  EndIf
  If nLin > 58
    Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
    nLin := 8
  EndIf
  @ nLin,000 PSAY __PrtFatLine()
  @ nLin++,001 PSAY ...
  @ nLin++,001 PSAY ...
  @ nLin++,001 PSAY ...
  @ nLin,000 PSAY __PrtThinLine()
  IncRegua()
  dbSkip()
End
IF nLin != 80
  nLin++
  If nLin > 60
    Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
  Endif
  @ nLin,000 PSAY __PrtThinLine()
  Roda( 0, cTitulo, cTamanho ) // Rodapé
Endif
If aReturn[5] = 1
  Set Printer To
  dbCommitAll()
  OURSPOOL( wrel )
Endif
MS_FLUSH()
Return

```

Pergunte

Inicializa as variáveis de pergunta (mv_par??).

Sintaxe

PERGUNTE(*cPergunta* , [*lPerg*] , [*cTitulo*]) -> lógico

Argumento	Obrigat.	Tipo	Descrição
<i>cPergunta</i>	Sim	C	Pergunta cadastrada no dicionário de dados (SX1) a ser utilizada.
<i>lPerg</i>	Não	L	Indica se exibirá a tela para edição.
<i>cTitulo</i>	Não	C	Título do diálogo.

Descrição

Esta função inicializa as variáveis de pergunta (mv_par01,...) baseado na pergunta cadastrado no Dicionário de Dados (SX1). Se o parâmetro lAsk não for especificado ou for verdadeiro será exibida a tela

para edição da pergunta e se o usuário confirmar as variáveis serão atualizadas e a pergunta no SX1 também será atualizada.

Exemplo

```
PERGUNTE("CON010")
```

Processa

Cria diálogo com uma régua de progressão.

Sintaxe

```
PROCESSA( bAcao, [ cTitulo ], [ cMsg ], [ lAborta] ) -> nil
```

Argumento	Obrigat.	Tipo	Descrição
bAcao	Sim	C	Função a ser executada.
cMsg	Não	C	Mensagem a ser exibida a baixo da régua de progressão
cTitulo	Não	C	Título de janela
lAborta	Não	L	Habilita botão cancelar.

Descrição

A função Processa() cria um diálogo onde a execução de um determinada função pode ser acompanhada através de uma régua de progressão. Para atribuir o valor total da régua utilizamos a função ProcRegua() e para incrementar a régua utilizamos a função IncProc().

Exemplo

```
User Function <nome-da-função>( )
Local bAcao := {|lFim| Exemplo(@lFim) }
Local cTitulo := ''
Local cMsg := 'Processando'
Local lAborta := .T.
PROCESSA( bAcao, cTitulo, cMsg, lAborta )
Return

Static Function Exemplo(lFim)
Local nI
ProcRegua(10000)
For nI := 1 To 10000
    If lFim
        Exit
    EndIf
    IncProc()
Next nI
```

Return

ProcRegua

Atribui o valor total da régua de progressão criada pela função Processa().

Sintaxe

PROCREGUA(*nTotal*) -> nil

Argumento	Obrigat.	Tipo	Descrição
nTotal	Sim	N	Valor total da régua.

Descrição

Após atribuir o valor total da régua, para incrementar utilizamos a função IncProc().

Exemplo

```
User Function <nome-da-função>( )
Local bAcao := { |lFim| Exemplo(@lFim) }
Local cTitulo := ''
Local cMsg := 'Processando'
Local lAborta := .T.
Processa( bAcao, cTitulo, cMsg, lAborta )
Return

Static Function Exemplo(lFim)
Local nI
PROCREGUA(10000)
For nI := 1 To 10000
    If lFim
        Exit
    EndIf
    IncProc()
Next nI
Return
```

PswAdmin

Verifica se um usuário pertence ao grupo de administradores.

Sintaxe

PSWADMIN(*cUsuario*, *cSenha*) -> numérico

Argumento	Obrigat.	Tipo	Descrição
cUsuario	Sim	C	Nome do usuário.

cSenha	Sim	C	Senha do usuário.
--------	-----	---	-------------------

Descrição

A função PswAdmin() retorna 0 (zero) se o usuário for do grupo de administradores, 1 (um) para usuário não administrador e 2 (dois) se for senha inválida.

Exemplo

```
User Function <nome-da-função>( cUsuario, cSenha )
Local lAdminst := .F.
Local nRet
nRet := PSWADMIN( cUsuario, cSenha )
If nRet == 0
    lAdminst := .T.
ElseIf nRet == 1
    APMsgInfo( "Usuário não é Administrador !" )
ElseIf nRet == 2
    APMsgInfo( "Senha invalida !" )
EndIf
Return lAdminst
```

PswID

Retorna o ID do usuário ou do grupo de usuário.

Sintaxe

PSWID() -> caracter

Descrição

Utilizada para retornar o ID do usuário ou do grupo de usuários após ter posicionado o arquivo de senha com a função PswSeek().

Exemplo

```
User Function <nome-da-função>( cUsuario )
Local cUserID
PswOrder(2)
If PswSeek( cUsuario, .T. )
    cUserId := PSWID() // Retorna o ID do usuário
EndIf
Return
```

PswName

Verifica senha de usuário.

Sintaxe

PSWNAME(*cSenha*) -> lógico

Argumento	Obrigat.	Tipo	Descrição
cSenha	Sim	C	Senha do usuário.

Descrição

A função PswName() verifica se a senha informada no parâmetro cSenha pertence ao usuário posicionado no arquivo de senha, se pertencer retornará verdadeiro(.T.).

Exemplo

```
User Function <nome-da-função>( cUsuario, cSenha )
Local lOk
PswOrder(2)
If PswSeek( cUsuario, .T. )
    lOk := PSWNAME( cSenha )
EndIf
Return lOk
```

PswOrder

Posiciona a ordem de indexação do arquivo de senhas.

Sintaxe

PSWORDER(*nOrdem*) -> nil

Argumento	Obrigat.	Tipo	Descrição
nOrdem	Sim	N	Ordem de indexação.

Descrição

A função PswOrder() posiciona a ordem de indexação de acordo com o parametro nOrdem, onde:

- 1 – ID;
- 2 – usuário;
- 3 – senha; e
- 4 – e-mail.

Exemplo

```
User Function <nome-da-função>( cUsuario )
Local cUserID
PSWORDER(2)
```

```
If PswSeek( cUsuario, .T. )
    cUserId := PswID() // Retorna o ID do usuário
EndIf
Return
```

PswRet

Retorna vetor contendo informações do Usuário ou do Grupo de Usuários.

Sintaxe

PSWRET() -> array

Descrição

A função PswRet() retorna dois tipos de vetores distintos, de acordo com o posicionamento do arquivo de senha. Se no segundo parâmetro da função PswSeek() for informado .T. a PswRet() retornará um vetor com informações do Usuários, caso contrário retornará informações do Grupo de Usuários.

Exemplo

```
// Exemplo 1
User Function <nome-da-função>( cUsuario )
Local aArray := {}
PswOrder(2)
If PswSeek( cUsuario, .T. )
    aArray := PSWRET() // Retorna vetor com informações do usuário
EndIf
Return
// Exemplo 2
User Function <nome-da-função>( cGrupoID )
Local cGrupo
PswOrder(1)
If PswSeek( cGrupoId, .F. )
    cGrupo := PSWRET()[1][2] // Retorna nome do Grupo de Usuário
EndIf
Return
```

PswSeek

Pesquisa e posiciona o arquivo de senhas.

Sintaxe

PSWSEEK(cID, [lUsuario]) -> lógico

Argumento	Obrigat.	Tipo	Descrição
cID	Sim	C	String contendo o ID do usuário ou do grupo de usuários.
lUsuario	Não	L	Se verdadeiro (.T.) pesquisa por usuários, se falso (.F.) pesquisa por grupo de usuários. O valor assumido como padrão é .T.

			grupo de usuários. O valor assumido como padrão é .T.
--	--	--	---

Descrição

A função PswSeek() pesquisa e posiciona o arquivo de senhas conforme a ordem de indexação e o parâmetro lUsuario, se encontrar o usuário ou grupo de usuários retorna verdadeiro (.T.).

Exemplo

```
User Function <nome-da-função>( cUsuario )
Local cUserID
PswOrder(2)// seleciona ordem de indexação
If PSWSEEK( cUsuario, .T. )
    cUserId := PswID() // Retorna o ID do usuário
EndIf
Return
```

ReadVar

Retorna o nome da variável que esta sendo editada.

Sintaxe

READVAR() -> caracter

Descrição

Retorna o nome da variável que esta sendo editada pela MsGetDados, MsGetDB e MsMGet (Enchoice) para ser usada na validação de um campo por exemplo.

Exemplo

```
cVar := READVAR()
If Type(cVar) == "C"
    APMsgInfo("Opção escolhida: "+&cVar)
EndIf
```

RetAcsName

Retorna a extensão do arquivo de acessos corrente.

Sintaxe

RETACSNAME() -> caracter

Descrição

A função RetAscName retorna a extensão do arquivo de acessos de acordo com idioma corrente.

Exemplo

```
Local cExt
cExt := RETACSNAME()
If 'ACS' $ cExt
    APMsgInfo( 'Menu Português' )
ElseIf 'ACE' $ cExt
    APMsgInfo( 'Menu Espanhol' )
ElseIf 'ACI' $ cExt
    APMsgInfo( 'Menu Inglês' )
EndIf
```

RetExtHlp

Retorna a extensão do help de campo.

Sintaxe

RETEXTHLP() -> caracter

Descrição

Esta função retorna uma string contendo a extensão do help de campos de acordo com idioma corrente.

Exemplo

```
Local cExt
cExt := RETEXTHLP()
If 'HLP' $ cExt
    APMsgInfo( 'Help de Campos em Português' )
ElseIf 'HLE' $ cExt
    APMsgInfo( 'Help de Campos em Espanhol' )
ElseIf 'HLI' $ cExt
    APMsgInfo( 'Help de Campos em Inglês' )
EndIf
```

RetExtHls

Retorna a extensão do help de soluções.

Sintaxe

RETEXTHLS() -> caracter

Descrição

Esta função retorna uma string contendo a extensão do help de soluções de acordo com idioma corrente.

Exemplo

```
Local cExt
```

```
cExt := RETEXTHLS()  
If 'HLS' $ cExt  
    APMsgInfo( 'Help de Soluções em Português' )  
ElseIf 'HSE' $ cExt  
    APMMsgInfo( 'Help de Soluções em Espanhol' )  
ElseIf 'HSI' $ cExt  
    APMMsgInfo( 'Help de Soluções em Inglês' )  
EndIf
```

RetExtHpr

Retorna a extensão do help do programa.

Sintaxe

RETEXTHPR() -> caracter

Descrição

Esta função retorna uma string contendo a extensão do help de programas de acordo com idioma corrente.

Exemplo

```
Local cExt  
cExt := RETEXTHPR()  
If 'HPR' $ cExt  
    APMsgInfo( 'Help de Programas em Português' )  
ElseIf 'HPE' $ cExt  
    APMMsgInfo( 'Help de Programas em Espanhol' )  
ElseIf 'HPI' $ cExt  
    APMMsgInfo( 'Help de Programas em Inglês' )  
EndIf
```

RetExtMnu

Retorna a extensão do menu corrente.

Sintaxe

RETEXTMNU() -> caracter

Descrição

Esta função retorna uma string contendo a extensão do menu de acordo com idioma corrente.

Exemplo

```
Local cExt  
cExt := RETEXTMNU()  
If 'MNU' $ cExt
```

```
APMsgInfo( 'Menu Português' )
ElseIf 'MNS' $ cExt
    APMsgInfo( 'Menu Espanhol' )
ElseIf 'MNE' $ cExt
    APMsgInfo( 'Menu Inglês' )
EndIf
```

RetFileName

Retorna o nome de um arquivo sem o caminho e sem a extensão.

Sintaxe

RETFILENAME(*cArquivo*) -> caracter

Argumento	Obrigat.	Tipo	Descrição
<i>cArquivo</i>	Sim	C	String contendo o nome do arquivo

Descrição

A função RetFileName() retorna o nome de um arquivo contido em uma string, ignorando o caminho e a extensão.

Exemplo

```
Local cArquivo := '\SIGAADV\ARQZZZ.DBF'
cArquivo := RETFILENAME( cArquivo )
// retorno 'ARQZZZ'
```

Roda

Imprime rodapé no relatório.

Sintaxe

RODA([*uPar1*], [*uPar2*], [*cTamanho*]) -> nil

Argumento	Obrigat.	Tipo	Descrição
<i>uPar1</i>	Não	U	Parâmetro reservado
<i>uPar2</i>	Não	U	Parâmetro reservado
<i>cTamanho</i>	Não	C	Tamanho do rodapé "P", "M", "G". Se o parâmetro não for informado, "M" será assumido como padrão.

Exemplo

```

User Function <nome-da-função>( )
Local cProgram := 'Exemplo' // nome do relatório
Local cAlias := 'XXX' // alias do arquivo
Local cPerg := 'XXXXXX' // grupo de perguntas
Local cTitulo := 'Titulo do relatório'
Local cDesc1 := 'Descrição'
Local cDesc2 := 'continuação da descrição'
Local cDesc3 := 'continuação da descrição'
Local lDic := .F. // não utiliza dicionário
Local aOrd := { '1º Ordem', '2º ordem', '3º ordem' }
Local lCompres := .F.
Local cTam := "G"
Private m_pag := 1
Private aReturn := { "Zebrado", 1,"Administracao", 1, 2, 1, "",1 }
Private wrel

Pergunte( cPerg, .F. )

wrel := SetPrint( cAlias, cProgram, cPerg, @cTitulo, cDesc1, cDesc2, cDesc3, lDic, aOrd ,
lCompres, cTam )
If nLastKey == 27
    Set Filter To
    Return
EndIf
SetDefault( aReturn, cAlias )
If nLastKey == 27
    Set Filter To
    Return
EndIf
RptStatus( { |lFim| Imprime( @lFim, cAlias, cTitulo, cProgram, cTamanho ) }, cTitulo )
Return

// rotina de impressão
Static Function Imprime( lFim, cAlias, cTitulo, cProgram, cTamanho )
Local nLin := 80
Local cCabec1 := 'Cabecalho 1'
Local cCabec2 := 'Cabecalho 2'
dbSelectArea( cAlias )
SetRegua( RecCount() )
While !Eof() .And. ( XX_COD >= MV_PAR01 .And. XX_COD <= MV_PAR02 )
    If lFim
        @Prow()+1,001 PSAY "CANCELADO PELO OPERADOR"
        Exit
    EndIf
    If nLin > 58
        Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
        nLin := 8
    EndIf
    @ nLin,000 PSAY __PrtFatLine()
    @ nLin++,001 PSAY ...
    @ nLin++,001 PSAY ...
    @ nLin++,001 PSAY ...
    @ nLin,000 PSAY __PrtThinLine()
    IncRegua()
    dbSkip()
End
IF nLin != 80
    nLin++
    If nLin > 60
        Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
    Endif
    @ nLin,000 PSAY __PrtThinLine()
    RODA( ,, cTamanho ) // Rodapé
EndIf
If aReturn[5] = 1
    Set Printer TO
    dbCommitAll()
    OurSpool( wrel )

```

```

Endif
MS_FLUSH()
Return

```

RptStatus

Cria um diálogo com régua de progressão.

Sintaxe

```
RPTSTATUS( bAcao, [ cTitulo ], [ cMsg ] ) -> nil
```

Argumento	Obrigat.	Tipo	Descrição
bAcao	Sim	C	Contendo a função a ser executada.
cTitulo	Não	C	Titulo da janela.
cMsg	Não	C	Mensagem a ser exibida a baixo da régua de progressão.

Descrição

A função RptStatus() exibe um diálogo onde a execução da função de relatório pode ser acompanhada através de uma régua de progressão. Para controlar a régua utilizamos as funções SetRegua() para atribuir o valor total da régua, e a função IncRegua() para incrementar. Caso o processamento seja interrompido através do botão cancelar, um valor lógico é retornado no parâmetro do código de bloco.

Exemplo

```

User Function <nome-da-função>( )
Local cProgram := 'Exemplo' // nome do relatório
Local cAlias := 'XXX' // alias do arquivo
Local cPerg := 'XXXXXX' // grupo de perguntas
Local cTitulo := 'Titulo do relatório'
Local cDesc1 := 'Descrição'
Local cDesc2 := 'continuação da descrição'
Local cDesc3 := 'continuação da descrição'
Local lDic := .F. // não utiliza dicionário
Local aOrd := { '1º Ordem', '2º ordem', '3º ordem' }
Local lCompres := .F.
Local cTam := "G"
Private m_pag := 1
Private aReturn := { "Zebrado", 1,"Administracao", 1, 2, 1, "",1 }
Private wrel
Pergunte( cPerg, .F. )
wrel := SetPrint( cAlias, cProgram, cPerg, @cTitulo, cDesc1, cDesc2, cDesc3, lDic, aOrd ,
lCompres, cTam )
If nLastKey == 27
    Set Filter To
    Return
EndIf
SetDefault( aReturn, cAlias )
If nLastKey == 27
    Set Filter To
    Return
EndIf

```

```

RPTSTATUS( { |lFim| Imprime( @lFim, cAlias, cTitulo, cProgram, cTamanho ) }, cTitulo )
Return

// rotina de impressão
Static Function Imprime( lFim, cAlias, cTitulo, cProgram, cTamanho )
Local nLin := 80
Local cCabec1 := 'Cabecalho 1'
Local cCabec2 := 'Cabecalho 2'
dbSelectArea( cAlias )
SetRegua( RecCount() )
While !Eof().And. ( XX_COD >= MV_PAR01 .And. XX_COD <= MV_PAR02 )
  If lFim
    @Prow()+1,001 PSAY "CANCELADO PELO OPERADOR"
    Exit
  EndIf
  If nLin > 58
    Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
    nLin := 8
  EndIf
  @ nLin,000 PSAY __PrtFatLine()
  @ nLin++,001 PSAY ...
  @ nLin++,001 PSAY ...
  @ nLin++,001 PSAY ...
  @ nLin,000 PSAY __PrtThinLine()
  IncRegua()
  dbSkip()
End
IF nLin != 80
  nLin++
  If nLin > 60
    Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
  Endif
  @ nLin,000 PSAY __PrtThinLine()
  Roda( 0, cTitulo, cTamanho ) // Rodapé
EndIf
If aReturn[5] = 1
  Set Printer TO
  dbCommitAll()
  OurSpool( wrel )
Endif
MS_FLUSH()
Return

```

SetDefault

Prepara ambiente de impressão.

Sintaxe

SETDEFAULT(aRetorno, [cAlias], [uPar1], [uPar2], [cTamanho], [nFormato]) -> nil

Argumento	Obrigat.	Tipo	Descrição
<i>aRetorno</i>	Sim	A	Configurações de impressão.
<i>cAlias</i>	Não	C	Alias do arquivo a ser impresso.
<i>uPar1</i>	Não	U	Parâmetro reservado.

<i>uPar2</i>	Não	U	Parâmetro reservado.
<i>cTamanho</i>	Não	C	Tamanho da página "P","M" ou "G"
<i>nFormato</i>	Não	N	Formato da página, 1 retrato e 2 paisagem.

Descrição

Após executar a função SetPrint() o vetor aReturn conterá as informações necessárias para que SetDefault() prepare o ambiente de impressão.

Exemplo

```
User Function <nome-da-função>( )
Local cProgram := 'Exemplo' // nome do relatório
Local cAlias := 'XXX' // alias do arquivo
Local cPerg := 'XXXXXX' // grupo de perguntas
Local cTitulo := 'Titulo do relatório'
Local cDesc1 := 'Descrição'
Local cDesc2 := 'continuação da descrição'
Local cDesc3 := 'continuação da descrição'
Local lDic := .F. // não utiliza dicionário
Local aOrd := { '1º Ordem', '2º ordem', '3º ordem' }
Local lCompres := .F.
Local cTam := "G"
Private m_pag := 1
Private aReturn := { "Zebrado", 1,"Administracao", 1, 2, 1, "",1 }
Private wrel

Pergunte( cPerg, .F. )

wrel := SetPrint( cAlias, cProgram, cPerg, @cTitulo, cDesc1, cDesc2, cDesc3, lDic, aOrd ,
lCompres, cTam )
If nLastKey == 27
    Set Filter To
    Return
EndIf
SETDEFAULT( aReturn, cAlias )
If nLastKey == 27
    Set Filter To
    Return
EndIf
RptStatus( {lFim| Imprime( @lFim, cAlias, cTitulo, cProgram, cTamanho ) }, cTitulo )
Return

// rotina de impressão
Static Function Imprime( lFim, cAlias, cTitulo, cProgram, cTamanho )
Local nLin := 80
Local cCabec1 := 'Cabecalho 1'
Local cCabec2 := 'Cabecalho 2'
dbSelectArea( cAlias )
SetRegua( RecCount() )
While !Eof().And. ( XX_COD >= MV_PAR01 .And. XX_COD <= MV_PAR02 )
    If lFim
        @Prow()+1,001 PSAY "CANCELADO PELO OPERADOR"
        Exit
    EndIf
    If nLin > 58
        Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
        nLin := 8
    EndIf
    @ nLin,000 PSAY __PrtFatLine()
    @ nLin++,001 PSAY ...
    @ nLin++,001 PSAY ...
    @ nLin++,001 PSAY ...
```

```

@ nLin,000 PSAY __PrtThinLine()
  IncRegua()
  dbSkip()
End
IF nLin != 80
  nLin++
  If nLin > 60
    Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
  Endif
  @ nLin,000 PSAY __PrtThinLine()
  Roda( 0, cTitulo, cTamanho ) // Rodapé
Endif
If aReturn[5] = 1
  Set Printer TO
  dbCommitAll()
  OurSpool( wrel )
Endif
MS_FLUSH()
Return

```

SetPrint

Interface onde são configuradas as opções de impressão.

Sintaxe

SETPRINT(*cAlias*, *cPrograma*, [*cPerg*], [*cTitulo*], [*cDesc1*], [*cDesc2*], [*cDesc3*], [*IDic*], [*aOrd*], [*ICompres*], [*cTam*], [*uPar1*], *IFiltro*, [*ICrystal*], [*cNomeDrv*], [*uPar2*], [*IServidor*], [*cPortalmpr*]) -> caracter

Argumento	Obrigat.	Tipo	Descrição
<i>cAlias</i>	Sim	C	Alias do arquivo a ser impresso.
<i>cPrograma</i>	Sim	C	Nome do arquivo a ser gerado em disco
<i>cPerg</i>	Não	C	Grupo de perguntas cadastrado no dicionário SX1.
<i>cTitulo</i>	Não	C	Título do relatório
<i>cDesc1</i>	Não	C	Descrição do relatório.
<i>cDesc2</i>	Não	C	Continuação da descrição do relatório.
<i>cDesc3</i>	Não	C	Continuação da descrição do relatório.
<i>IDic</i>	Não	L	Para impressão de cadastro genérico permite a escolha dos campos a serem impressos.
<i>aOrd</i>	Não	A	Ordem(s) de impressão.
<i>ICompres</i>	Não	L	Se verdadeiro (.T.) habilita escolha o formato da impressão.
<i>cTam</i>	Não	C	Tamanho do relatório "P", "M" ou "G".

<i>uPar1</i>	Não	U	Parâmetro reservado
<i>IFiltro</i>	Não	L	Se verdadeiro (.T.) permite a utilização do assistente de filtro.
<i>ICrystal</i>	Não	L	Se verdadeiro (.T.) permite integração com Crystal Report.
<i>cNomeDrv</i>	Não	C	Nome de um driver de impressão.
<i>uPar2</i>	Não	U	Parâmetro reservado.
<i>IServidor</i>	Não	L	Se verdadeiro (.T.) força impressão no servidor.
<i>cPortalImpr</i>	Não	C	Define uma porta de impressão padrão.

Descrição

A função SetPrint() cria a interface (diálogo) 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(). Após confirmada, os dados são armazenados no vetor *aReturn* que será passado como parâmetro para função SetDefault().

(*) controla o número de páginas.

(**) vetor contendo as opções de impressão, sua estrutura é composta de 8 (oito) elementos: 1º - caracter, tipo do formulário; 2º - numérico, opção de margem; 3º - caracter, destinatário; 4º - numérico, formato da impressão; 5º - numérico, dispositivo de impressão; 6º - reservado; 7º - reservado; 8º - numérico, ordem.

Exemplo

```
User Function <nome-da-função>( )
Local cProgram := 'Exemplo' // nome do relatório
Local cAlias := 'XXX' // alias do arquivo
Local cPerg := 'XXXXXX' // grupo de perguntas
Local cTitulo := 'Titulo do relatório'
Local cDesc1 := 'Descrição'
Local cDesc2 := 'continuação da descrição'
Local cDesc3 := 'continuação da descrição'
Local lDic := .F. // não utiliza dicionário
Local aOrd := { '1º Ordem', '2º ordem', '3º ordem' }
Local lCompres := .F.
Local cTam := "G"
Private m_pag := 1
Private aReturn := { "Zebrado", 1,"Administracao", 1, 2, 1, "",1 }
Private wrel

Pergunte( cPerg, .F. )
wrel := SETPRINT( cAlias, cProgram, cPerg, @cTitulo, cDesc1, cDesc2, cDesc3, lDic, aOrd ,
lCompres, cTam )
If nLastKey == 27
    Set Filter To
    Return
EndIf
SetDefault( aReturn, cAlias )
If nLastKey == 27
    Set Filter To
    Return
EndIf
RptStatus( { |lFim| Imprime( @lFim, cAlias, cTitulo, cProgram, cTamanho ) }, cTitulo )
Return

// rotina de impressão
Static Function Imprime( lFim, cAlias, cTitulo, cProgram, cTamanho )
Local nLin := 80
Local cCabec1 := 'Cabecalho 1'
```

```

Local cCabec2 := 'Cabecalho 2'
dbSelectArea( cAlias )
SetRegua( RecCount() )
While !Eof() .And. ( XX_COD >= MV_PAR01 .And. XX_COD <= MV_PAR02 )
  If lFim
    @Prow()+1,001 PSAY "CANCELADO PELO OPERADOR"
    Exit
  EndIf
  If nLin > 58
    Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
    nLin := 8
  EndIf
  @ nLin,000 PSAY __PrtFatLine()
  @ nLin++,001 PSAY ...
  @ nLin++,001 PSAY ...
  @ nLin++,001 PSAY ...
  @ nLin,000 PSAY __PrtThinLine()
  IncRegua()
  dbSkip()
End
IF nLin != 80
  nLin++
  If nLin > 60
    Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
  Endif
  @ nLin,000 PSAY __PrtThinLine()
  Roda( 0, cTitulo, cTamanho ) // Rodapé
EndIf
If aReturn[5] = 1
  Set Printer TO
  dbCommitAll()
  OurSpool( wrel )
Endif
MS_FLUSH()
Return

```

SetRegua

Atribui valor total da régua de progressão.

Sintaxe

SETREGUA(*nTotal*) -> nil

Argumento	Obrigat.	Tipo	Descrição
<i>nTotal</i>	Sim	N	Valor total da régua.

Descrição

Após executar a função RptStatus() é necessário atribuir o valor total da sua régua, isso será feito através da função SetRegua().

Exemplo

```

User Function <nome-da-função>( )
Local cProgram := 'Exemplo' // nome do relatório

```

```

Local cAlias := 'XXX' // alias do arquivo
Local cPerg := 'XXXXXX' // grupo de perguntas
Local cTitulo := 'Titulo do relatório'
Local cDesc1 := 'Descrição'
Local cDesc2 := 'continuação da descrição'
Local cDesc3 := 'continuação da descrição'
Local lDic := .F. // não utiliza dicionário
Local aOrd := { '1º Ordem', '2º ordem', '3º ordem' }
Local lCompres := .F.
Local cTam := "G"
Private m_pag := 1
Private aReturn := { "Zebrado", 1, "Administracao", 1, 2, 1, "", 1 }
Private wrel

Pergunte( cPerg, .F. )

wrel := SetPrint( cAlias, cProgram, cPerg, @cTitulo, cDesc1, cDesc2, cDesc3, lDic, aOrd ,
lCompres, cTam )
If nLastKey == 27
    Set Filter To
    Return
EndIf
SetDefault( aReturn, cAlias )
If nLastKey == 27
    Set Filter To
    Return
EndIf
RptStatus( { |lFim| Imprime( @lFim, cAlias, cTitulo, cProgram, cTamanho ) }, cTitulo )
Return

// rotina de impressão
Static Function Imprime( lFim, cAlias, cTitulo, cProgram, cTamanho )
Local nLin := 80
Local cCabec1 := 'Cabecalho 1'
Local cCabec2 := 'Cabecalho 2'
dbSelectArea( cAlias )
SETREGUA( RecCount() )
While !Eof() .And. ( XX_COD >= MV_PAR01 .And. XX_COD <= MV_PAR02 )
    If lFim
        @Prow()+1,001 PSAY "CANCELADO PELO OPERADOR"
        Exit
    EndIf
    If nLin > 58
        Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
        nLin := 8
    EndIf
    @ nLin,000 PSAY __PrtFatLine()
    @ nLin++,001 PSAY ...
    @ nLin++,001 PSAY ...
    @ nLin++,001 PSAY ...
    @ nLin,000 PSAY __PrtThinLine()
    IncRegua()
    dbSkip()
End
IF nLin != 80
    nLin++
    If nLin > 60
        Cabec( cTitulo, cCabec1, cCabec2, cProgram, cTamanho, 15 ) // cabeçalho
    Endif
    @ nLin,000 PSAY __PrtThinLine()
    Roda( 0, cTitulo, cTamanho ) // Rodapé
EndIf
If aReturn[5] = 1
    Set Printer TO
    dbCommitAll()
    OurSpool( wrel )
Endif
MS_FLUSH()
Return

```

SixDescricao

Retorna descrição de uma chave de índice.

Sintaxe

SIXDESCRICA() -> caracter

Descrição

Esta função retorna a descrição da chave de índice, de acordo com o registro posicionado no SIX e idioma corrente.

Exemplo

```
User Function <nome-da-função>( cChave, cOrdem )
Local cSixDesc := ""
dbSelectArea( "SIX" )
dbSetOrder( 1 )
If dbSeek( cChave+ cOrdem )
    cSixDesc := SIXDESCRICA()
EndIf
Return
```

VerSenha

Verifica se o usuário tem acesso a determinada opção.

Sintaxe

VERSENHA(nOpc) -> lógico

Argumento	Obrigat.	Tipo	Descrição
nOpc	Sim	N	Número da opção.

Descrição

Esta função verifica se o usuário corrente tem acesso a determinada opção que foi cadastrada no Configurador nas propriedades do usuário ("Excluir produto", "Alterar produto", ...).

Exemplo

```
If !VERSENHA(36) //verifica se o usuário pode alterar a data base do sistema.
    Return
EndIf
```

X1Def01

Retorna o conteúdo da primeira definição da pergunta (caso seja combo).

Sintaxe

X1DEF01() -> caracter

Descrição

O conteúdo retornado pela função será de acordo com o registro posicionado no SX1 e o 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
```

X1Def02

Retorna o conteúdo da segunda definição da pergunta (caso seja combo).

Sintaxe

X1DEF02() -> caracter

Descrição

O conteúdo retornado pela função será de acordo com o registro posicionado no SX1 e o 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
```

X1Def03

Retorna o conteúdo da terceira definição da pergunta (caso seja combo).

Sintaxe

X1DEF03() -> caracter

Descrição

O conteúdo retornado pela função será de acordo com o registro posicionado no SX1 e o 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
```

X1Def04

Retorna o conteúdo da quarta definição da pergunta (caso seja combo).

Sintaxe

X1DEF04() -> caracter

Descrição

O conteúdo retornado pela função será de acordo com o registro posicionado no SX1 e o 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
```

X1Def05

Retorna o conteúdo da quinta definição da pergunta (caso seja combo).

Sintaxe

X1DEF05() -> caracter

Descrição

O conteúdo retornado pela função será de acordo com o registro posicionado no SX1 e o idioma corrente.

Exemplo

```
User Function <nome-da-função>( cGrupo, cPerg )
Local cDef01
Local cDef02
Local cDef03
Local cDef04
Local cDef05
dbSelectArea("SX1")
dbSetOrder(1)
If dbSeek( cGrupo + cPerg ) // grupo da pergunta + o numero da perg.
    cDef01 := X1Def01()
    cDef02 := X1Def02()
    cDef03 := X1Def03()
    cDef04 := X1Def04()
    cDef05 := X1DEF05()
EndIf
Return
```

X1Pergunt

Retorna descrição de uma pergunta contida no dicionário de dados SX1.

Sintaxe

X1PERGUNT() -> caracter

Descrição

O conteúdo retornado pela função será de acordo com o registro posicionado no SX1 e o 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

Retorna o descrição de uma tabela contida no dicionário de dados SX2.

Sintaxe

X2NOME() -> caracter

Descrição

O conteúdo retornado pela função será de acordo com o registro posicionado no SX2 e o idioma corrente.

Exemplo

```
User Function <nome-da-função>( )
Local cTabela
dbSelectArea( "SX2" )
dbSetOrder(1)
If dbSeek( "SA1" )
    cTabela := X2NOME()
EndIf
Return
```


X3CBox

Retorna o conteúdo de um campo tipo combo contido no dicionário de dados SX3

Sintaxe

X3CBOX() -> variável

Descrição

Esta função retorna conteúdo do campo combo de acordo com o registro posicionado no SX3 e o 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

Retorna o descrição de um campo contido no dicionário de dados SX3

Sintaxe

X3DESCRIC() -> caracter

Descrição

Esta função retorna a descrição do campo de acordo com o registro posicionando no SX3 e o idioma corrente.

Exemplo

```
User Function <nome-da-função>( )
Local cTitulo
Local cDescri
Local cCombo
dbSelectArea( "SX3" )
dbSetOrder(2)
If dbSeek( cCampo )
    cTitulo := X3Titulo()
    cDescri := X3DESCRIC()
    cCombo := X3Cbox()
EndIf
Return
```

```
EndIf  
Return
```

X3Picture

Retorna a mascara de um campo do dicionário de dados SX3.

Sintaxe

X3PICTURE(*cCampo*) -> caracter

Argumento	Obrigat.	Tipo	Descrição
<i>cCampo</i>	Sim	C	Nome de um campo cadastrado no SX3

Exemplo

```
User Function <nome-da-função>( cCampo )  
Local cPicture  
cPicture := X3PICTURE( cCampo )  
Return cPicture
```

X3Titulo

Retorna o título de um campo contido no dicionário de dados SX3

Sintaxe

X3TITULO() -> caracter

Descrição

Esta função retorna o título do campo de acordo com o registro posicionado no SX3 e o idioma corrente.

Exemplo

```
User Function <nome-da-função>( )  
Local cTitulo  
dbSelectArea( "SX3" )  
dbSetOrder( 2 )  
If dbSeek( "A1_COD" )  
    cTitulo := X3TITULO()  
EndIf  
Return
```

X3Uso

Verifica se o campo está disponível para uso.

Sintaxe

X3USO(*cUsado*, [*nModulo*]) -> lógico

Argumento	Obrigat.	Tipo	Descrição
cUsado	Sim	C	Conteúdo do campo X3_USADO a ser pesquisado
nModulo	Não	N	Numero do módulo, caso não seja informado será assumido como padrão o número do módulo corrente.

Descrição

Esta função retornará um valor lógico, se for uma campo usado verdadeiro (.T.), caso contrário falso (.F.).

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

Retorna a descrição de uma tabela cadastrada no dicionário de dados SX5

Sintaxe

X5DESCRI() -> caracter

Descrição

Esta função retorna a descrição da tabela de acordo com o registro posicionado no SX5 e o idioma corrente.

Exemplo

```
User Function <nome-da-função>( cFilial, cTabela, cChave )  
Local cDescr  
dbSelectArea("SX5")  
dbSetOrder(1)  
If dbSeek( cFilial+cTabela+cChave )  
    cDescr := X5DESCRI ()  
EndIf  
Return
```

X6Conteud

Retorna o conteúdo do parâmetro cadastrado no dicionário de dados SX6.

Sintaxe

X6CONTEUD() -> caracter

Descrição

Esta função retorna o conteúdo de um parâmetro de acordo com o registro posicionado no SX6 e idioma corrente.

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
```

X6Desc1

Retorna o primeiro complemento da descrição de um parâmetro cadastrado no dicionário de dados SX6.

Sintaxe

X6DESC1() -> caracter

Descrição

Esta função retorna o conteúdo do primeiro complemento da descrição de um parametro de acordo com o registro posicionado no SX6 e o idioma corrente.

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
```

X6Desc2

Retorna o segundo complemento da descrição de um parâmetro cadastrado no dicionário de dados SX6.

Sintaxe

X6DESC2() -> caracter

Descrição

Esta função retorna o conteúdo do segundo complemento da descrição de um parâmetro de acordo com o registro posicionado no SX6 e o idioma corrente.

Exemplo

```
User Function <nome-da-função>( cFilial, cParam )
Local cDescr
Local cConteud
dbSelectArea("SX6")
dbSetOrder(1)
If dbSeek( cFilial+cParm )
    cDescr    := X6Descric()
    cDescr    += X6Desc1()
    cDescr    += X6DESC2()
    cConteud  := X6Conteud()
EndIf
Return
```

X6Descric

Retorna a descrição de um parâmetro cadastrado no dicionário de dados SX6.

Sintaxe

X6DESCRIC() -> caracter

Descrição

Esta função retorna o conteúdo da descrição de um parâmetro de acordo com o registro posicionado no SX6 e o idioma corrente.

Exemplo

```
User Function <nome-da-função>( cFilial, cParam )
Local cDescr
Local cConteud
dbSelectArea("SX6")
dbSetOrder(1)
If dbSeek( cFilial+cParm )
    cDescr := X6DESCRIC()
    cDescr += X6Desc1()
    cDescr += X6Desc2()
    cConteud := X6Conteud()
EndIf
Return
```

XADescric

Retorna a descrição dos folders (pastas) cadastrados no dicionário de dados SXA.

Sintaxe

XADESCRIC() -> caracter

Descrição

Esta função retorna o conteúdo da descrição dos folders de acordo com o registro posicionado no SXA e o 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 := XADESCRIC()
EndIf
Return
```

XBDescr

Retorna descrição de uma consulta SXB.

Sintaxe

XBDESCRI() -> caracter

Descrição

Esta função retorna o conteúdo da descrição de uma consulta de acordo com o registro posicionado no SXB e o idioma corrente.

Exemplo

```
User Function <nome-da-função>( cAlias )
Local cDescr
dbSelectArea( "SXB" )
dbSetOrder(1)
If dbSeek( cAlias + "1" )
    cDescr := XBDESCRI ( )
EndIf
Return
```

Instalação de Serviços AP6 e AP5 no mesmo Servido

- 1-Tire do ar o AP5 Server que está como console (CTRL-BREAK);
- 2-Pare o serviço do AP6 Server que está automático no Win2000 Server;
- 3-Remova o serviço do AP6 Server chamado pelo iniciar / executar o AP6SVR com a cláusula "-remove"

- 4-Inclua no INI do AP5SVR a cláusula

```
[Service]
name=AP5
Displayname=AP5
```

- 5-Inclua no INI do AP6SVR a cláusula

```
[Service]
name=AP6
Displayname=AP6
```

6-Suba o serviço do AP6SVR a partir de iniciar / executar com a cláusula "-install"

7-Suba o serviço do AP5SVR a partir de iniciar / executar com a cláusula "-install"

Se subir legal os dois, confirme pelo service do Win2000 que ambos estão no ar, startados e como automáticos.

Se não der certo, volte à situação original, ou seja, AP6SVR automático como serviço e AP5SVR como console.

Integração Siga Advanced - Office

Através do Office Kit, encontrado no CD de instalação da versão 407 em diante, o Siga oferece integração com os aplicativos do pacote Office da Microsoft. É importante lembrar que o Siga Advanced possui integração somente com o Excel e que a integração com o Word está disponível somente para o módulo de Controle de Documentos do Siga Quality Celerina.

Instalação

Para instalar o SIGA Advanced for Windows com Microsoft Excel, o usuário deve possuir:

- Ø Hardware – Configuração mínima: Pentium 133 com 16 MB RAM, recomendado Pentium 166 com 32 MB RAM.
- Ø Microsoft Excel – A versão mínima é 7.0 (95), recomendado o uso da versão 8.0 (97)

1. Execute o programa de instalação do SIGA Advanced.
2. Instale o SIGA Office Kit nas estações.

Observações

1. A instalação do Office Kit deve ser feita em todas as estações que irão utilizar a integração.
2. A MICROSIGA envia dois arquivos exemplos de Planilha Excel, ADVDEMO e ADVDEMO7, para serem utilizados nas versões Excel 97 e Excel 95, respectivamente. Estes arquivos são instalados no diretório X:\Arquivos de Programa \ SIGA Advanced Office Kit \ Samples (onde X: é drive onde está instalado o SIGA Advanced).

Utilizando a Integração

Pode-se acessar o Microsoft Excel a partir de qualquer Módulo SIGA Advanced, com exceção do SIGACFG (Configurador).

1. Escolha a opção "Planilha Excel" no menu Miscelânea.
Se for o primeiro acesso ao Microsoft Excel através do Siga Advanced, será solicitado o diretório onde está o executável do Microsoft Excel, essa configuração será gravada no arquivo "ADV97BRW.INI". Esse processo deve ser realizado em cada estação que for utilizar o SIGA Advanced for Windows com o Microsoft Excel.
2. Clique sobre o botão "OK" para iniciar o Microsoft Excel. Após a inicialização do Microsoft Excel, a conexão pode ser cancelada através do SIGA Advanced, clicando no botão "Cancelar", ou fechando o Microsoft Excel.

OBS: A integração somente funcionará se o Excel for aberto a partir do Siga Advanced.

Usando as funções do Siga no Excel

Existem 4 formas de se utilizar as funções do SIGA Advanced em uma planilha Excel. Veja a seguir:

1. Este procedimento é recomendado para usuários menos experientes, pois o Microsoft Excel abre uma janela assistente para que sejam informados os parâmetros, facilitando o uso de referência a outras células.
 - Clique na célula que deve receber o resultado da função SIGA Advanced e selecione a opção "Funções" no menu Inserir.
 - No lado esquerdo da janela são relacionadas as categorias de funções disponíveis. Selecione a categoria de funções "Definidas pelo usuário".
 - Na caixa de listagem "Nome da Função", selecione a função que será utilizada. Será apresentada uma janela Assistente, onde devem ser informados os parâmetros da função, conforme solicitado e definido pela sua sintaxe.
 - Informe os parâmetros da função e clique no botão "OK" para finalizar.
2. Outra forma de se utilizar as funções SIGA, é digitando a função diretamente na célula. Esta forma não conta com o assistente, e portanto, o usuário deve conhecer a sintaxe da função para informar corretamente seus parâmetros (recomendada apenas para usuários mais experientes). Para a informação de uma fórmula no Excel, deve-se digitar o sinal de igual "=".
3. A terceira forma é utilizar a função SIGA() para identificar o início da informação de funções SIGA Advanced. Para tanto, deve ser usada a seguinte sintaxe:
SIGA("nome da função"; 1º parâmetro da função; 2º parâmetro da função;...)
Por exemplo, informando:
=SIGA("MesExtenso";2) Desta forma, será retornado o mês "Fevereiro".
Observe que o nome da função SIGA Advanced é informado entre aspas. (" ").
4. A quarta forma, também é iniciar a informação com a função Siga(), porém as funções SIGA Advanced, juntamente com seus parâmetros, devem ser digitados entre aspas, obedecendo a sintaxe padrão que utiliza os sinais de parênteses. Por exemplo:
=SIGA("MesExtenso(2)")

Cuidados com a utilização de funções

O Microsoft Excel utiliza um padrão de data diferente do SIGA Advanced, por esse motivo qualquer função de planilha do Advanced utilizada a partir do menu "Inserir Fórmula" do Microsoft Excel ou da sintaxe =Siga("nomefunção"; Par01;Par02) que utilize data como parâmetro ou retorne uma data, deve obedecer as regras abaixo:

- Ø Formatar a célula como data, para as funções que retornem data.
- Ø Utilizar como parâmetro das funções do SIGA Advanced uma referência de célula cujo conteúdo seja uma data, ou digitar a data no formato do Microsoft Excel "00:00 10/05/98".

Toda função do SIGA Advanced utilizada a partir do menu "Inserir Fórmula" do Microsoft Excel ou da sintaxe "=Siga("nomefunção";Par01;Par02)" que utilize valor lógico (.T./F.) como parâmetro, deve obedecer o padrão do Microsoft Excel (atenção! com o idioma do Microsoft Excel):

- Ø .T. - VERDADEIRO/TRUE
- Ø .F. - FALSO/FALSE

Se ao executar uma função SIGA Advanced, o Excel retornar o erro "#VALOR" ou a mensagem "As funções do SIGA Advanced não constam na lista de funções definidas pelo usuário", verifique os seguintes tópicos para solucionar:

- Ø Tenha certeza de que o Excel foi iniciado a partir do SIGA Advanced.

- Ø Através da opção "Suplementos" do Menu "Ferramentas" do Excel, utilize o botão "Procurar" para localizar o "Add In" do SIGA Advanced (AdvXla80.XLA para a versão 8.0 ou AdvXla para versão 7.0 do Excel, que estão no diretório de Bibliotecas do Office).

ExecBlocks

Pode-se utilizar Execblock nas células do Microsoft Excel com algumas restrições:

- Ø Não devem ser utilizadas entradas de dados ou qualquer outro tipo de "Tela".
- Ø O Execblock deve obrigatoriamente retornar um valor.
- Ø O Execblock deve ser pequeno para evitar a perda da conexão do SIGA Advanced com o Microsoft Excel.
- Ø A variável que recebe o parâmetro passado pelo Excel é sempre a PARAMIXB, que deve ser tratada dentro do execblock. Caso seja necessário utilizar mais de um parâmetro no programa, ele já deve vir do Excel concatenado em uma única String.
- Ø Sintaxe: =EXECBLOCK("Nome da função",PAR1)
- Ø O Rdmake utilizado no ExecBlock deve estar dentro do diretório Sigaadv da base que está sendo utilizada para chamar o Excel e já estar compilado.

Funções Genéricas

POSICIONE

Posiciona o arquivo em um determinado registro.
Sintaxe: Posicione(ExpC1, ExpN1, ExpC2, ExpC3)

onde:

ExpC1 - Alias do arquivo

ExpN1 - Ordem utilizada

ExpC2 - Chave pesquisa

ExpC3 - Campo a ser retornado

Ex.: Posicione("SA1",1,xFilial("SA1")+001,"A1_NOME")

MEDIA

Calcula a taxa média de uma determinada moeda em um mês específico.

Sintaxe: Media (ExpN1, ExpN2, ExpN3)

onde:

ExpN1 - Moeda a ser calculada a média

ExpN2 - Mês a ser calculado

ExpN3 - Ano a ser calculado

Ex.:

Media (2,10,94)

Caso o mês e o ano não sejam passados como parâmetros serão assumidos o mês e ano da database.

SOMAR

Faz o somatório de um arquivo, retornando o valor acumulado de um campo determinado.

Sintaxe: Somar(ExpC1, ExpC2, ExpC3)

onde:

ExpC1 - Alias do arquivo

ExpC2 - Condição para soma

ExpC3 - Campo a ser somado

Ex.:

Somar("SI1","I1_CLASSE='S' ", "I1_SALANT")

Caso o usuário não deseje definir nenhuma condição, a ExpC2 deve ser ".T.".

GRUPO

Retorna o somatório das células pertencentes a um determinado grupo.

Sintaxe: Grupo(ExpC1)

onde:

ExpC1 - Grupo a ser somado

Ex.: Grupo(1)

CONTAR

Conta o número de registros de acordo com a condição determinada.

Sintaxe: Contar(ExpC1, ExpC2)

onde:

ExpC1 - Alias do arquivo

ExpC2 - Condição para a contagem

Ex:

Contar("SC1","C1_DATPRF < dDataBase")

Funções Contábeis

SALDO

Devolve o saldo de uma determinada conta.

Sintaxe: Saldo(ExpC1, ExpN1,ExpN2)

onde:

ExpC1 - Código da Conta

ExpN1 - Período do saldo desejado

ExpN2 - Moeda desejada para obtenção do saldo.

Ex.: Saldo("11101",12,1)

MOVIMENTO

Devolve o movimento (débito-crédito) de uma determinada conta.

Sintaxe: Movimento(ExpC1,ExpN1,ExpN2,ExpD1)

onde:

ExpC1 - Código da conta

ExpN1 - Mês do movimento desejado

ExpN2 - Moeda desejada para obtenção do movimento

ExpD1 - Data do exercício desejado

Ex.:

Movimento("43",1,1, CTOD ("01/01/95")) :

retorna o 1º período

Movimento("43",1,1,CTOD("01/01/96")) :

retorna o 13º período

DEBITO

Devolve o valor a débito de uma determinada conta.

Sintaxe: Debito(ExpC1, ExpN1, ExpN2)

onde:

ExpC1 - Código da Conta

ExpN1 - Mês do movimento desejado

ExpN2 - Moeda desejada para obtenção do valor a débito

Ex.: Debito("11103",03,1)

CREDITO

Devolve o valor a crédito de uma conta.

Sintaxe: Credito(ExpC1, ExpN1, ExpN2)

onde:

ExpC1 - Código da Conta

ExpN1 - Mês do movimento desejado

ExpN2 - Moeda desejada para obtenção do valor a crédito

Ex.: Credito("11103",03,1)

SOMAMOVIM

Retorna o movimento dentro de um intervalo de contas analíticas

Sintaxe: SomaMovim(ExpC1, ExpC2, ExpN1, ExpN2)

onde:

ExpC1 - Código da conta De

ExpC2 - Código da conta Até

ExpN1 - Referente ao mês

Exp N2 - Moeda desejada para obtenção do valor

Ex.: SomaMovim("41304","41305",12,1)

SOMASALDO

Retorna o saldo atual entre um intervalo de contas.

Sintaxe SomaSaldo(ExpC1, ExpC2, ExpN1, ExpN2)

onde:

ExpC1 - Código da conta De

ExpC2 - Código da conta Até

ExpN1 - Referente ao período

ExpN2 - Moeda desejada para obtenção do valor

Ex.: SomaSaldo("31001","31010",12,1)

ORÇADO

Retorna o valor orçado de uma conta..

Sintaxe Orcado(ExpC1, ExpN1, ExpN2, ExpD1)

onde:

ExpC1 - Código da conta De

ExpN1 - Referente ao período

ExpN2 - Moeda desejada para obtenção do valor

ExpD1 - Data para conversão (em formato data ou character), caso não informa-da, será utilizada a DataBase do sistema.

Ex.: Orcado("11102001",1,1)

MESEXTENSO

Devolve o nome do mês por extenso.

Sintaxe: MesExtenso(ExpX1)

onde:

ExpX1 - Poderá ser enviado o número de um mês, tanto do tipo caracter como numérico, ou poderá ser enviada uma data. Caso nada seja informado, será devolvido o nome do mês referente a Data Base.

Ex.: MesExtenso(01)

VARIAÇÃO

Retorna a variação em percentual entre dois valores.

Sintaxe: Variacao(ExpN1, ExpN2)

onde:

ExpN1 - Primeiro fator comparativo

ExpN2 - Segundo fator comparativo

Ex.: Variacao(100000,50000)

MOVIMCC

Retorna o movimento de um centro de custo mais conta contábil (extracontábil).

Sintaxe: MovimCC(ExpC1, ExpC2, ExpN1, ExpN2)

onde:

ExpC1 - Código do centro de custo

ExpC2 - Código da conta contábil

ExpN1 - Referente ao mês

ExpN2 - Moeda desejada para obtenção do valor

Ex.:

MovimCC("3001", 111001,Month(Ddatabase),1)

ORCADOCC

Recupera o valor orçado da Conta x Centro de Custo para utilização na planilha.

Sintaxe: OrcadoCC(ExpC1,ExpC2,ExpN1,ExpN2,ExpD1),

onde:

ExpC1 - Conta

ExpC2 - Centro de Custo

ExpN1 - Período (default mês da database)

ExpN2 - Moeda (default 1)

ExpD1 - Data para conversão se moeda de 2 a 5 - (default dDataBase).

Ex.: OrcadoCC("111001","3001",3,2)

SUMMOVIMCC

Retorna o movimento de um intervalo de centro de custos extracontábeis. Poderá ser parametrizados também um grupo de contas.

Sintaxe:

SumMovimCC(ExpC1,ExpC2,ExpC3,ExpC4,ExpN1,ExpN2)

onde:

ExpC1 - do Centro de Custo

ExpC2 - até Centro de Custo

ExpC3 - da Conta

ExpC4 - até a Conta

ExpN1 - Mês (default mês da database)

ExpN2 - Moeda (default 1)

Ex.:

SumMovimCC(3001","3100","31001","31010",12,1)

GRAVAORCADO

Permite que um determinado valor calculado pela planilha seja gravado no Arquivo de Orçamentos.

Sintaxe:

GravaOrcado(ExpC1,ExpN1,ExpN2,ExpN3)

onde:

ExpC1 - Conta Contábil a ser orçada

ExpN1 - Número da célula onde o valor estará contido

ExpN2 - Mês a ser orçado (se nulo, será mês corrente)

ExpN3 - Moeda a ser orçada (se nula, será moeda nacional)

Por exemplo: Obtém-se um valor na célula "022" referente à conta "11102001", sendo que este deverá ser orçado na contabilidade para o mês "07" e na moeda "1". Para tanto, cria-se em outra célula a seguinte fórmula:

GravaOrcado("11102001",#022,7,1)

Uma grande vantagem deste recurso é que caso a planilha seja recalculada e o valor alterado, o valor orçado será automaticamente atualizado. Esta função irá devolver à planilha o conteúdo "<<< Orçado >>>"

Para inibir a impressão destas células, basta acessar as suas propriedades e desabilitar a visibilidade.

SALDOCUSTO

Calcula o saldo dos centro de custos extracontábeis.

Sintaxe:

SaldoCusto(ExpC1,ExpC2,ExpC3,ExpC4,ExpN1,ExpN2)

onde:

ExpC1 - Centro de Custo Inicial

ExpC2 - Centro de Custo Final

ExpC3 - Conta Inicial

ExpC4 - Conta Final

ExpN1 - Mês (se nula, assume o mês de referência da database)

ExpN2 - Moeda (se nula, será assumido 1)

Ex.: SaldoCusto("1007", "1099", "3", "4", "10, 1)

SOMACONTAS

Retorna o saldo acumulado de um grupo de contas, de acordo com a sintaxe apresentada. Esta função considera somente contas de classe "A" – analítica

Sintaxe: SomaContas(ExpC1,ExpN1,ExpN2)

onde:

ExpC1 - Lista de contas, cercada por aspas ("").

O Separador ":" (dois pontos) forma intervalo de contas De - Até. O separador "," (vírgula) , informa separação de contas.

ExpN1 - Mês (default mês da database)

ExpN2 - Moeda (default 1)

Exemplos:

SomaContas("11101001",3,1)

Devolve o saldo da conta em questão no mês 3 na moeda 1.

SomaContas("11101001,11101002,11102237")

Devolve a soma dos saldos das contas em questão.

SomaContas("11101001:11102,31101001")

Devolve a soma dos saldos das contas 11101001 até 11102 mais o saldo da conta 31101001.

Funções Financeiras

SLDBCO

Retorna o saldo bancário em uma data.

Sintaxe: SldBco(ExpC1,ExpC2,ExpC3,ExpD1,ExpN1)

onde:

ExpC1 - Código do Banco

ExpC2 - Agência Bancária

ExpC3 - Conta Bancária

ExpD1 - Data do Saldo

ExpN1 - Moeda do Saldo Bancário

Ex.: SldBco("409","00198","011122", dDataBase,1)

SLDRECEBER

Retorna o saldo a receber em uma data.

Sintaxe: SldReceber(ExpD1,ExpN1,ExpL1)

onde:

ExpD1 - Data do Movimento a Receber.

ExpN1 - Moeda - Default 1

ExpL1 - Se .T. Até a Data, .F. Somente Data (o padrão é .T.)

Ex.: SldReceber(Data,1,.T.)

SLDPAGAR

Retorna o saldo a pagar em uma determinada data.

Sintaxe: SldPagar(ExpD1,ExpN1,ExpL1)

onde:

ExpD1 - Data do Movimento a Pagar

(padrão é dDataBase)

ExpN1 - Moeda (padrão é 1)

ExpL1 - Se .T. Até a Data, .F. Somente Data (padrão é .T.)

Ex.: SldPagar(dDataBase,1,.T.)

SLDCLIENTE

Retorna o saldo a receber do cliente em uma determinada data.

Sintaxe: SldCliente(ExpC1,ExpD1,ExpN1,ExpL1)

onde:

ExpC1 - Cliente+Loja

ExpD1 - Data do Movimento a Receber

(padrão é dDataBase)

ExpN1 - Moeda

ExpL1 - Se .T. considera o saldo do SE5

(padrão é .T.)

Ex.: SldCliente("00000101",dDataBase)

SLDFORNECE

Retorna o saldo a pagar do fornecedor em uma data

Sintaxe: SldFornece(ExpC1,ExpD1,ExpN1,ExpL1)

onde:

ExpC1 - Fornecedor+Loja

ExpD1 - Data do Movimento a Pagar (padrão é dDataBase)

ExpN1 - Moeda - (padrão é 1)

ExpL1 - Se .T. considera o saldo do SE5

(padrão é .T.)

Ex.: SldFornece("00000101")

FINNATPRV

Retorna o valor previsto de cada natureza.

Sintaxe: FinatPrv(ExpC1,ExpD1,ExpD2,ExpN1)

onde:

ExpC1 - Natureza a ser Pesquisada

ExpD1 - Data Inicial para cálculo

ExpD2 - Data Final de cálculo

ExpN1 - Moeda de Saída

Ex.:

FinNatPrv("R001",CtoD("01/01/98"),dDataBase,1)

FINNATREA

Retorna o valor realizado da Natureza.

Sintaxe: FinNatRea(ExpC1,ExpD1,ExpD2,ExpN1)

onde:

ExpC1 - Natureza a ser Pesquisada

ExpD1 - Data Inicial para cálculo

ExpD2 - Data Final de cálculo

ExpN1 - Moeda de Saída

Ex.: FinNatRea("R001",CtoD("01/01/98"),dDataBase,1)

FINNATORC

Retorna o valor orçado da natureza.

Sintaxe: FinNatOrc(ExpC1,ExpN1,ExpN2)

onde:

ExpC1 - Natureza a ser Pesquisada

ExpN1 - Mês para cálculo

ExpN2 - Moeda de Saída

Ex.: FinNatOrc("R001",10,1)

ROTEIRO PARA INSTALAÇÃO DO TOPCONNECT COM SQL/SERVER 7.0 MICROSOFT :

- O Windows NT 4.0 deve ser da versão em inglês e Ter o service Pack atualizado(caso possua estações Workstation as recomendações de service pack atualizado deve ser aplicado na estação).
- Na instalação do SQL de ser selecionada a opção Custom Character Set/sort order/Unicode Collation/Character Set: 1252/ISO CHARACTER SET (DEFAULT)
- SORT ORDER: BINARY ORDER E UNICODE COLLATION/LOCALE IDENTIFIER: BINARY ORDER
- No "Query Analyzer"(Ferramenta do SQL/Server 7.0) executar SP_HELPSORT, para verificar se o SQL, está Configurado como Binary Order ("Sort Order = 50, bin_iso_"), caso não esteja DEVERÁ SER REFEITA A INSTALAÇÃO DO SQL SERVER.
- Acessar a Ferramenta Enterprise Manager do Microsoft SQL/Server 7.0, no servidor SQL destinado para o Siga, abrir a árvore indicada pelo sinal de "+" ao lado esquerdo deste Server.
- Abrir a pasta Databases, clique com o botão direito sobre qualquer Database e escolha New Database. Onde o Nome deste Database, por exemplo, será DADOSADV, cria-lo com o tamanho previamente definido e demais parametros default , e em TRANSACTIONS LOG será definido a área destinada a Log e o ideal que seja em outro disco (NÃO OUTRA PARTIÇÃO) diferente do que estiver armazenado o DATABASE,

o tamanho também deverá ser previamente definido. Ex.: DATABASE 500MB e em TRANSACTION LOG 150MB.

- Em PROPERTIES/OPTIONS do DATABASE a opção "*Truncat Log on Checkpoint*", tem a função de estratégia de Backup, se a empresa optar por efetuar Backup's de Log, tal opção poderá ficar desmarcada.
- Agora entre em Security e em Logins, e com o botão direito sobre qualquer outro login, escolha New Login, seguindo o exemplo, Nome = SIGA, ativar o SQL Server authentication Password = SIGA, Server Roles marcar a opção SYSTEM ADMINISTRATORS e em DATABASE ACCESS atribuir permissões (permit) para o database criado, no caso DADOSADV.
- *A parte do SQL Server está pronta. Agora resta apenas a configuração do TOPConnect Manager e do SigaAdv.*
- Instalar o TOPConnect, e após o termino do Install, encerrar todos os serviço e reiniciar o Servidor. Caso o processo seja de Atualização do TOPConnect 1 para TOPConnect 2, deve ser efetuada a remoção do TOPConnect 1(e o que for referente ao topconnect) através de Adicionar e Remover Programas que está no Painel de Controle do Windows NT, antes de Instalar o TOPConnect 2.
- Topconnect 2 / Em Propriedades de Ambiente. Configuração do ODBC SQL Server Setup : Name = TOPSQL, SERVER=(nome do servidor) [Caso o TOPconnect esteja no mesmo servidor que o SQL/Server deve-se manter a opção LOCAL – Desta forma a conexão entre o TOPConnect e o SQL não estará trafegando na Rede], NEXT>.
- Marcar a opções ☐ With SQL Server authentication using a login ID e ☐ Connect to SQL Server to obtain default settings Login ID : SIGA , Password :SIGA. NEXT>
- Marcar.: ☐ Change the default database to: (Selecionar o Database criado para o Siga) e desmarcar as opções ☐ Use ANSI quoted e ☐ Use ANSI nulls, paddings and warnings . NEXT>.
- Próxima tela marcar ☐ Perform Translation for character data, e os demais parametros serão marcados deacordo com a política de administração do Banco.
- Selecionar na Barra de Ferramentas do Topconnect o ícone = a uma Chave (ou Editar Licenças), para que possa ser detectado o nº de série, da máquina e assim entrar em contato com o suporte Topconnect (CLIENTE DEVERÁ ENTRAR EM CONTATO COM O ATENDIMENTO DO DISK-SIGA.: (0xx11)3981-7100)para obter a chave de acesso conforme a quantidade de usuários adquiridos.
- SIGA ADVANCED 4.07 : No diretório Sigaadv\WINSQL, existe um arquivo chamado ADV97.INI. Edite-o, os parâmetros corretos para o SigaAdv com SQL Server são: Database = MSSQL7, Alias = TOPSQL, Contype = NPIPE, Server = NOME do servidor NT com o TOPConnect.
- AP5 : Localizar em AP5\BIN o arquivo AP5SRV.ini e edita-lo, neste arquivo deverá ser configurado a clausula [TOPCONNECT] como segue o exemplo conforme o padrão adotado neste roteiro:

```
[TopConnect]
DataBase=MSSQL7
Server= *nome do servidor ou o endereço IP do servidor
ALIAS=TOPSQL
Contype=NPIPE ou Contype=TCPIP *dependendo da opção do Server.
```
- Caso seja necessário a migração de base DBF para o Banco verifique, antes de executar o DBF2SQL, o DBF2SQL.INI. O Alias tem que ser igual à MSSQL7/TOPSQL, neste exemplo.
- Está pronta toda sua configuração.
- CASO seja necessário que o" SQL SERVER deva estar em UM SERVIDOR NT E TOPCONNECT EM OUTRO NT".

- No 1º Instalar o SQL SERVER. No 2º instalar o client do SQL e o topconnect, e apontar o client para a 1ª máquina, o TOPCONNECT deve apontar para o SQL Server (1ª máquina) o ADV97.INI deverá indicar o nome do servidor da 2ª máquina, aonde o Topconnect está instalado.
- <ftp://ftp.microsiga.com.br/topconnect/nt> - Download Instalador do TOPConnect_2.

1.1. Ambiente Desejável para uma instalação de 25 usuários:

Servidor Windows NT 4.0 com TopConnect, Banco de dados
MsSql/Oracle/Informix/..., SigaAdvanced/AP5.

- Memória = 512 MB ou mais
- Discos = 2 discos SCSI com 9GB de 80mbit/s com dois adaptadores para garantir acesso simultaneo dos discos. Para base de dados grandes (com mais de 1GB de dados) recomendamos um Disk Array do tipo Raid-5 de 4 Discos de 9GB com memória de 256 a 512 para cache de leitura.
- Processador = 2 processador Pentium III Xeon 500mhz com 512 de cache.
- Rede = Adaptador de 100 mbits, utilizar swite(não cascadear Hubs)

ROTEIRO PARA INSTALAÇÃO DO TOPCONNECT MSSQL 2000

- *Processo de instalação do SQL 2000 – Criação de Instance.*
 1. *SQL SERVER 2000 COMPONENTS*
 2. *INSTALL DATABASE SERVER*
 - *Local Server*
 - *Create a New Instance of SQL Server or Install Client Tools*
 - *Server and Client Tools*
 - *A proxima opção irá definir a criação de uma Instance para isto desmarque a opção Default., caso permaneça marcado, será criado uma instace como será no SQL 7.0.*
 - *Na proxima opção selecionar a forma Custom, que permitirar selecionar o local para a instalação do banci e marcar os componentes a serem instalados.*
 - *SERVICES ACCOUNTS*
Selecionar a forma que o Sql ira Startar.

Ex: Service Settings

Use The Local System Account

- AUTHENTICATION MODE

Windows Authentication Mode

- COLLATION SETTINGS

Sql collations

Binary order, for use with the 437 (U.S.English) Character Set.

- NETWORK LIBRARIES <next> START COPYING FILES <next>

- *Segue o resultado do Query Analyzer ao executar o sp_helpsort : Latin1-General, binary sort for Unicode Data, SQL Server Sort Order 30 on Code Page 437 for non-Unicode Data*

- Acessar a Ferramenta Enterprise Manager do Microsoft SQL/Server 2000, no servidor SQL destinado para o Siga, abrir a árvore indicada pelo sinal de "+" ao lado esquerdo deste Server.
- Abrir a pasta Databases, clique com o botão direito sobre qualquer Database e escolha New Database. Onde o Nome deste Database, por exemplo, será DADOSADV, cria-lo com o tamanho previamente definido e demais parametros default , e em TRANSACTIONS LOG será definido a área destinada a Log e o ideal que seja em outro disco (NÃO OUTRA PARTIÇÃO) diferente do que estiver armazenado o DATABASE, o tamanho também deverá ser previamente definido. Ex.: DATABASE 500MB e em TRANSACTION LOG 150MB.
- Em PROPERTIES/OPTIONS do DATABASE a opção "*Truncat Log on Checkpoint*", tem a função de estratégia de Backup, se a empresa optar por efetuar Backup's de Log, tal opção poderá ficar desmarcada.
- Agora entre em Security e em Logins, e com o botão direito sobre qualquer outro login, escolha New Login, seguindo o exemplo, Nome = SIGA, ativar o SQL Server authentication Password = SIGA, Server Roles marcar a opção SYSTEM ADMINISTRATORS e em DATABASE ACCESS atribuir permissões (permit) para o database criado, no caso DADOSADV.
- *A parte do SQL Server está pronta. Agora resta apenas a configuração do TOPConnect Manager e do SigaAdv.*
- Instalar o TOPConnect, e após o termino do Install, encerrar todos os serviço e reiniciar o Servidor, Caso o processo seja de Atualização do TOPConnect 1 para TOPConnect 2, deve ser efetuada a remoção do TOPConnect 1(e o que for referente ao topconnect) através de Adicionar e Remover Programas que está no Painel de Controle do Windows NT, antes de Instalar o TOPConnect 2.
- Topconnect 2 / Em Propriedades de Ambiente. Configuração do ODBC SQL Server Setup : *Name* = TOPSQL, *SERVER*=(nome do servidor) [Caso o TOPconnect esteja no mesmo servidor que o SQL/Server deve-se manter a opção LOCAL – Desta forma a conexão entre o TOPConnect e o SQL não estará trafegando na Rede], *NEXT*>.
- Marcar a opções ☐ With SQL Server authentication using a login ID e ☐ Connect to SQL Server to obtain default settings Login ID : SIGA , Password : SIGA. *NEXT*>
- Marcar.: ☐ Change the default database to: (Selecionar o Database criado para o Siga) e desmarcar as opções ☐ Use ANSI quoted e ☐ Use ANSI nulls, paddings and warnings . *NEXT*>.
- Próxima tela marcar ☐ Perform Translation for character data, e os demais parametros serão marcados deacordo com a política de administração do Banco.
- Selecionar na Barra de Ferramentas do Topconnect o ícone = a uma Chave (ou Editar Licenças), para que possa ser detectado o nº de série, da máquina e assim entrar em contato com o suporte Topconnect

(CLIENTE DEVERÁ ENTRAR EM CONTATO COM O ATENDIMENTO DO DISK-SIGA.: (0xx11)3981-7100) para obter a chave de acesso conforme a quantidade de usuários adquiridos.

- CLIENTE SIGAADVANCED

- No diretório AP5\BIN, existe um arquivo chamado AP5SRV.INI. Edite-o, os parâmetros corretos para O PROTHEUS com SQL Server são: Database = MSSQL7, Alias = TOPSQL, Contype = NPIPE, Server = NOME do servidor NT com o TOPConnect.
- <ftp://ftp.microsiga.com.br/topconnect/nt> - Download de executáveis para TOPConnect e das atualizações do TOPConnect_2.

MsExecAuto

Objetivo

Fazer manutenção automática (inclusão, alteração e exclusão) das rotinas de manipulação de dados do sistema, automatizando o processo de entrada de dados sem a necessidade de desenvolver rotinas específicas.

Aplicação

Esta técnica é aplicada em todas as versões Protheus, porém esta documentação baseou-se na versões superiores a AP6.09.

Vantagens

- 1) Interface : Os dados de entrada são enviados a rotina em forma de campos e conteúdos (array) e desta forma não é necessário a apresentação de nenhuma interface ao usuário.
- 2) Segurança : A utilização de rotinas automáticas aumenta consideravelmente a segurança do sistema, uma vez que utiliza as validações padrões e diminui os problemas causados por atualização de versão ou inclusão de customizações nas rotinas padrões do sistema.
- 3) Agilidade no processo : Aumenta consideravelmente o tempo de desenvolvimento das customizações que necessitam de entrada de dados. Exemplo: Importação de pedido de venda.

Procedimentos

Existem duas maneiras de utilizar a rotina automática, sendo elas:

1. Sem Interface
2. Com Interface

Para a utilizacao da rotina automatica sem interface deve-se, configurar o ambiente utilizando-se o comando PREPARE ENVIRONMENT e chamar diretamente o nome da função.

Exemplo:

User Function IncProd()

```
Local aRotAuto := {}
Local nOpc := 3 // inclusao
Private IMsHelpAuto := .t. // se .t. direciona as mensagens de help para o arq. de log
Private IMsErroAuto := .f. //necessario a criacao, pois sera //atualizado quando
houver
//alguma inconsistencia nos parametros
```

PREPARE ENVIRONMENT EMPRESA '99' FILIAL '01' MODULO 'FAT'

```
Begin Transaction
aRotAuto:= {{ 'B1_COD' , '1010' , Nil} , ;
{ 'B1_DESC' , 'Produto teste' , Nil} , ;
{ 'B1_TIPO' , 'PA' , Nil} , ;
{ 'B1_UM' , 'UN' , Nil} , ;
{ 'B1_LOCPAD' , '01' , Nil} , ;
{ 'B1_PICM' , 'O' , Nil} , ;
{ 'B1_IPI' , 'O' , Nil} , ;
{ 'B1_PRV1' , '100' , Nil} , ;
{ 'B1_LOCALIZ' , 'N' , Nil} , ;
{ 'B1_CODBAR' , '789888800001' , Nil} }
```

```
MSExecAuto({ |x,y| mata010(x,y)} , aProduto, nOpc)
```

```
If IMsErroAuto
```

```
DisarmTransaction()
```

```
break
```

```
EndIf
```

```
End Transaction
```

```
If IMsErroAuto
```

```
/*
```

Se estiver em uma aplicao normal e ocorrer alguma inconsistencia nos parametros passados, mostrar na tela o log informando qual coluna teve a inconsistencia.

```
*/
```

```
Mostraerro()
```

```
Return .f.
```

```
EndIf
```

```
Return .t.
```

Ja para rotinas que possuem interface, devemos usar a MSExecAuto, pois a mesma tem a função de guardar o ambiente , executar a função automatica e retornar de onde parou.

Exemplo:

User Function IncProd()

```

Local aRotAuto := {}
Local nOpc := 3 // inclusao
Private IMsHelpAuto := .t. // se .t. direciona as mensagens de help para o arq. de log
Private IMsErroAuto := .f. //necessario a criacao, pois sera //atualizado quando
houver
//alguma incosistencia nos parametros
Begin Transaction
aRotAuto:= {{ 'B1_COD' , '1010' , Nil} , ;
{ 'B1_DESC' , 'Produto teste' , Nil} , ;
{ 'B1_TIPO' , 'PA' , Nil} , ;
{ 'B1_UM' , 'UN' , Nil} , ;
{ 'B1_LOCPAD' , '01' , Nil} , ;
{ 'B1_PICM' , 0 , Nil} , ;
{ 'B1_IPI' , 0 , Nil} , ;
{ 'B1_PRV1' , 100 , Nil} , ;
{ 'B1_LOCALIZ' , 'N' , Nil} , ;
{ 'B1_CODBAR' , '789888800001' , Nil} }

MSExecAuto({ |x,y| mata010(x,y) } , aProduto , nOpc)
If IMsErroAuto
DisarmTransaction()
break
EndIf
End Transaction
If IMsErroAuto
/*
Se estiver em uma aplicao normal e ocorrer alguma incosistencia nos parametros
passados,mostrar na tela o log informando qual coluna teve a incosistencia.
*/
Mostraerro()
Return .f.
EndIf
Return .t.

```

RELAÇÃO DE ERROS DO TOPCONNECT

Constant Name: TCF_NoError

Value: 0

Meaning: A Chamada da Função não foi completada como desejada.

Constant Name: TCF_NoRouterInstalled

Value: -1

Meaning: Você está tentando executar a função TC_Connect sem ter uma camada de comunicação.

Recovery: Você necessita ter um Appc ou TCP/IP router disponível

Constant Name: TCF_NoConnection

Value: -2

Meaning: Você está tentando executar uma função sem estabelecer conexão previa.

Recovery: Execute a função do TC_Connect

Constant Name: TC_NoUserSecurity

Value: -4

Meaning: Você necessita fornecer um user+password válido com o TC_Setuser

Recovery:

Constant Name: TCF_No_More_Connections

Value: -6

Meaning: Não Há mais conexões disponíveis.

Recovery: Feche uma das conexões e tente nova conexão.

Constant Name: TCF_ScanTableError

Value: -7

Meaning: Há uma diferença entre o formato do registro corrente que o AS/400 arquiva e o formato digitado na PC Table.

Recovery: Obtenha o registro do formato AS/400 para o arquivo usando no programa do TClient, e check a tabela

Constant Name: TCF_InvalidFile

Value: -10

Meaning: Você está tentando abrir um arquivo que não pode ser encontrado

Recovery: Verifique se o arquivo está na biblioteca indicada sobre abertura

Constant Name: TCF_UnknownFile

Value: -11

Meaning: Você está tentando abrir um arquivo que não está associado com um processo no AS/400.

Recovery: Verifique o identifier de tabela utilizado em abertura.

Constant Name: TCF_InvalidProgram

Value: -12

Meaning: O programa especificado na função do TC_Call não pode ser encontrado.

Recovery: Verifique a localização do Nome do programa.

Constant Name: TCF_InvalidOperation

Value: -13

Meaning: O programa está utilizando uma função inválida.

Recovery: Cheque a existência da função.

Constant Name: TCF_InvalidKeyNum

Value: -14

Meaning: Você tem especificado um número de campo chave maior que o arquivo, como uma função de acesso chave.

Recovery: Veja o valor de parâmetro do n_keys

Constant Name: TCF_TooManyFiles

Value: -16

Meaning: Você está tentando abrir um arquivo, mas o número de máximo de arquivos ao servidor excedeu.

Recovery: Feche um dos arquivos abertos e tente abrir o arquivo outra vez.

Constant Name: TCF_InvalidNumRecs

Value: -17

Meaning: Um número negativo está especificado no parâmetro do lock_or_count da operação multiple-read

Recovery: Verifique o valor de parâmetro do Lock_or_count da operação multiple-read.

Constant Name: TCF_CallFailed

Value: -18

Meaning: O programa que chamou a função do TC_Call falhou.

Recovery: Consulte no AS/400 as mensagens de erro.

Constant Name: TCF_CommandFailed

Value: -19

Meaning: O comando do OS/400 chamado na função do TC_Call Falhou.

Recovery: Consulte no AS/400 as mensagens de erro.

Constant Name: TCF_OverrideFailed

Value: -20

Meaning: O OVRDBF comando de OS/400 está falhando na abertura o arquivo.

Recovery: Consulte no AS/400 as mensagens de erro.

Constant Name: TCF_QueryFailed

Value: -21

Meaning: Um erro foi produzido no OPNQRYF ou no comando SELECT do AS/400

Recovery: Consulte no AS/400 as mensagens de erro.

Constant Name: TCF_OpenFailed

Value: -23

Meaning: O arquivo não pode ser aberto.

Recovery: Consulte no AS/400 as mensagens de erro.

Constant Name: TCF_NotOpened

Value: -24

Meaning: Você está tentando uma operação com um arquivo que não está aberto.

Recovery: Verifique se o arquivo foi aberto apropriadamente.

Constant Name: TCF_NoRecordFound

Value: -25

Meaning: O registro solicitado não pode ser encontrado como uma função de acesso ao campo chave.

Recovery: Verifique os valores do campo chave no registro.

Constant Name: TCF_EndOfrecords

Value: -26

Meaning: Você chegou no fim do arquivo ou o fim de registros que estão em Read_Multiple.

Recovery: Esta é uma mensagem consultiva

Constant Name: TCF_NoWritePossible

Value: -27

Meaning: Você não pode incluir um novo registro no arquivo.

Recovery: Verifique o valor da chave de índice em registro. Você pode estar tentando escrever um registro com uma chave duplicada ou database está cheio.

Constant Name: TCF_NoRecordEqual

Value: -28

Meaning: Não Há registro pertencendo à chave especificada na função do TCF_ReadE.

Recovery: Verifique o valor das chaves de índice em registro.

Constant Name: TCF_UpdateFailed

Value: -29

Meaning: Você não pode criar registro no arquivo.

Recovery: Verifique a chave utilizada na criação, e assegure que o registro não esteja Locado.

Constant Name: TCF_DeleteFailed

Value: -30

Meaning: Você não pode deletar o registro no arquivo.

Recovery: Verifique a chave utilizada, e assegure que o registro não está Locado.

Constant Name: TCF_RecordLocked

Value: -31

Meaning: O registro está locado por outra tarefa.

Recovery: Verifique qual a tarefa que não está liberando o registro

Constant Name: TCF_NoAuthorization

Value: -33

Meaning: O uso do TOPconnect não está autorizado.

Recovery: Isto indica que a conexão com o AS/400 não pode ser completada porque a chave do TOPConnect é inválida ou está expirada.

Constant Name: TCF_TooManyUsers

Value: -34

Meaning: O número de máximo de usuários conectados foi alcançado.

Recovery: Alguns usuários deverão desconectar ou mais licenças serão necessitadas.

Constant Name: TCF_NoDBConnection

Value: -35

Meaning: O Database não pode ser acessado.

Recovery:

Value: -58

Meaning: "Não Há Memória Do DOS"

Recovery: Indica que não pode ter o desempenhado necessario devido insuficiência de memória corrente para a situação.

Value: -90

Meaning: " Memória Insuficiente"

Recovery: Indica que o trabalho não pode ser feito devido a insuficiência de Memória do PC . Corrija a situação.

Hardlock

1-) Qual o objetivo do "Hardlock" ?

Juntamente com a versão 7.10 a Microsiga disponibiliza um hardware de proteção que tem por objetivo a segurança das informações usando algoritmos de criptografia (recurso para proteção de mensagens eletrônicas) baseados em chaves de 128 bits, liberando mecanismos mais eficientes para proteger o sistema contra acessos indevidos. Aliado a isso, mais agilidade no processo de liberação de senhas e ganho de performance nos processos de controle aos acessos simultâneos (semaforização) são obtidos.

2-) O cliente recebeu a versão 7.10 mas ainda não recebeu o "Hardlock" o que deve-se fazer ?

Isto não irá ocorrer, pois o "pacote" da versão AP7 contempla:

- 1 CD Instalação AP7 Windows
- 1 CD Instalação AP7 Linux
- 1 "Hardlock" – Paralela
- 1 "Bottom" AP7
- 1 Recibo de entrega da versão, a ser assinado pelo cliente

Portanto, tendo o recibo assinado, comprova o recebimento de todos os itens acima !

3-) Com quem devo falar na Microsiga para solicitar a contra senha do "Hardlock" ?

O próprio Departamento de Senhas é o responsável pela liberação e envio das contra-senhas do "Hardlock".

4-) Continua existindo a necessidade de liberação de senha do sigamat.emp ?

Sim. O Objetivo da liberação de senhas no Sigamat é de controlar o número de empresas usuárias do sistema, enquanto o "Hardlock" é responsável pela criptografia dos dados, assim como o controle de usuários simultâneos.

5-) Como realizar testes em clientes sem utilizar as licenças oficiais ?

Os testes a partir da versão 7.10 ficam sendo possíveis, sem consumo de licenças do ambiente oficial, somente na empresa "99 – Teste / Matriz ", ou utilizando-se da senha de emergência.

6-) Como fica a empresa teste ?

A empresa "99 - Teste" não requer a utilização nem a autenticação através do "Hardlock", portanto, seu uso está liberado. O número de registros que antes era limitado a 50 passa a não existir mais, sendo este substituído pelo controle de uso por no máximo 2 usuários simultâneos.

7-) Como ficam as liberações no caso em que o "Hardlock", depois de colocado em uso, falhar?

Para este caso existe o procedimento de liberação da senha de emergência, que neste caso, irá liberar o uso do sistema até que a substituição ou manutenção seja realizada. Em caso de falha constatada no dispositivo, este será substituído gratuitamente no 1º ano e ao custo de tabela da Store (Hoje em US\$ 43,55) a partir de então.

8-) Quantos "Hardlock" o cliente precisa utilizar em seus ambientes ?

Somente um, pois o cliente irá eleger qual "servidor" Protheus também será o servidor de licenças, e este por sua vez poderá autenticar todos os demais servidores da empresa.

9-) O cliente perdeu o "Hardlock", e agora?

Caso o cliente tenha perdido ou tenha furtado o "Hardlock" ele deverá providenciar um Boletim de ocorrência e comprar a nova chave de segurança pelo valor de US\$ 43,55

10-) O cliente modificou seu ambiente de hardware, ou seja, a máquina quebrou instalou novamente, mudou para uma máquina nova, etc. O mesmo "Hardlock" funcionará?

Irá funcionar normalmente desde que seja instalado novamente, e reconfigurado o AP7SRV.INI, caso o nome / IP da máquina tenha sido alterado.

11-) Onde e como deve ser instalado o "Hardlock" ?

Na porta paralela ou USB, de acordo com a disponibilidade do Servidor do cliente. O próprio instalador do AP7 irá solicitar informações quanto a porta e o nome do servidor em que será instalado o "Hardlock".

12-) Posso um ambiente com três "sites" distintos, onde dois deles estão centralizados e o terceiro está trabalhando de forma independente ? Como este ambiente deverá ser com a utilização do "Hardlock" ?

Contratualmente a Microsiga exige um contrato para cada localidade (física), portanto, existindo este caso, o cliente receberá 2 "Hardlock"'s, 1 para ser usado nos que são centralizados, e um adicional para que seja usado na localidade independente. Cabe salientar que a Microsiga fornece gratuitamente 1 "Hardlock" para cada contrato de manutenção vigente e caso exista apenas um contrato de manutenção, o cliente deverá adquirir mais um "Hardlock" pelo preço de US\$ 43,55

13-) O Cliente recebeu o "Hardlock" paralelo e somente tem saída USB ?

O mesmo deve entrar em contato com a unidade de Atendimento e relacionamento da Microsiga para que seja providenciada a sua substituição.

14-) Existe vantagem de performance entre o "Hardlock" paralelo e ou USB ?

Tecnicamente sim, pois as portas USB's são em média 5x mais rápidas que as paralelas, porém isto não implicara em maior ou menor performance do sistema, pois a comunicação com a chave de segurança se dá em raras ocasiões.

15-) Quando devo utilizar a senha emergencial ?

Quando por algum motivo qualquer o "Hardlock" , ou a porta em que ele está instalado venha a falhar, portanto, até sua manutenção, o cliente deverá utilizar a senha emergencial.

No caso da necessidade de se montar um ambiente teste exatamente com as características do cliente também deve ser solicitada a senha de emergência.

16-) O "Hardlock" não funciona mesmo com a contra senha correta, o que devo fazer?

Verifique a configuração da porta (paralela ou USB). Realize testes utilizando um outro servidor para ter certeza se o problema é ou não no Hardlock. Cabe salientar que todos os "HardLocks" são testados na Microsiga no momento da liberação.

17-) Caso o cliente seja um "DataCenter " e tenha um número de licenças que deva ser distribuído para cada um de seus clientes, o que fazer ?

Existem duas alternativas para resolver este problema :

17.1 -> Distribuir o licenciamento em "n" "HardLocks" e "n" servidores

17.2 -> Ter um único "HardLock" e controlar os acessos através do ponto de entrada "CHKEXEC" além da utilização da chave "SEMAFOROKEY" no ambiente ("Environment" corrente) que está configurado no AP7SRV.INI

18-) Como configurar o servidor de licenças ?

O próprio instalador do AP7 solicita as informações necessárias durante este processo para se utilizar o "Hardlock". No arquivo .INI do Servidor do AP7, haverão duas sessões conforme abaixo:

```
[LICENSESERVER]
enable=1
port=porta-do-license-server

[LICENSECLIENT]
server=nome-do-servidor
```

port=porta-do-license-server

19-) Preciso um servidor de licença para cada servidor de processamento do Protheus?

Não. O cliente tendo 2 ou mais servidores de aplicação, o mesmo irá eleger um deles para ser o servidor de licenças, onde então será instalado o "Hardlock", e a partir deste momento, os demais passarão a pedir autenticação para ele.

20-) Podemos ter mais de um "HardLock" no mesmo servidor ?

Sim, mas apenas um da Microsiga.

16 Programando com Schedule de Relatorios

Abrangência

Como o sistema permite que a emissão de relatórios possa ser programada (schedule) é fundamental que se utilize as rotinas padrões para a emissão dos mesmo. O controle do schedule é feito pela função SetPrint. Sendo assim, não é suportado interface com data entry durante o processo de relatório, visto que isto inviabilizará a utilização do mesmo. A não ser em relatórios específicos e que sejam inviáveis a utilização de shedule (ex. Impressão de Cheques) este procedimento deverá ser adotado corretamente.

Caso exista alguma entrada de dados que seja possível ser assumida qualquer valor apenas no schedule deve-se adotar o seguinte procedimento :

Usar a variável __cInternet, que se estiver com valor .T. (Verdadeiro) estamos no processo de schedule.

Instruções para instalação das Storeds Procedures

1. Arquivos anexados:
 - a. .sps - Arquivo encryptado com as Storeds Procedures.
 - b. .txt - Arquivo que documenta o conteúdo dos arquivos .sps.
 - c. .pat - Patch do Protheus. (é necessário solicitar)
2. Instalando o .pat (*se houver*).

Abra o IDE do Protheus, selecione o menu *Ferramentas -> Aplicação de Patches*. Escolha o arquivo .pat e clique em Ok. Este arquivo deve estar dentro do sever do protheus, preferencialmente dentro da pasta Sigadv.

Obs.: Para baixar o patch, pode acessar :
<ftp://microsiga.com.br/Ap7/Procedures>

Este patch contém os seguintes arquivos:
CFGX051.PRW, MATA280.PRX, MATA300.PRX

Cada solicitante tem a responsabilidade de validar os Patches antes de aplicá-lo na área de produção do cliente. Como procedimento obrigatório, deve-se efetuar um backup da situação atual, antes da aplicação do Patch.

Devemos estar ciente que o Patch completo que esta no nosso site FTP, contém todas as alterações efetuadas até a data especificada pelo laboratório de produtos da Microsiga, portanto devemos observar que a atualização do RPO ou Patch completo deste site, não garante que os Patches emergenciais aplicados estarão contemplados. Recomendamos que em caso de dúvida,verifiquem nos documentos existentes no FTP

de sua versão as datas e alterações das rotinas realizadas junto com o arquivo texto anexo, se as mesmas não existirem, reapliquem os Patches emergenciais ou façam nova solicitação dos mesmos.

Somente atenderemos as solicitações originadas de e-mails com domínio @microsig.com.br, e faremos o envio para esta mesma conta, o solicitante deverá ser sempre o canal de redirecionamento ao cliente.

3. Instalando o arquivo .sps.

Entre no módulo configurador e selecione a opção do instalador de procedures, normalmente encontrada no menu *Ambiente -> Cadastros -> Stored Procedures*.

Aparecerá uma tela para que sejam selecionadas as filiais para as quais as Stored Procedures serão instaladas. Selecione as filiais desejadas e clique em ok.

Caso não exista o menu, crie. A função que chama o configurador é a CFGX051().

Variáveis Públicas Disponíveis no Protheus

- `dDataBase (D, 8)` à Contém a data selecionada na entrada do sistema.
- `cUsuario (C, 533)` à Contém informações do usuário:
 - o De 001 a 006 – Senha do Usuário (6 dígitos)
 - o De 007 a 021 – Nome do Usuário (15 dígitos)
 - o De 022 a 533 – Permissões de Acesso (512 dígitos)
- `cSenha (C, 6)` à Senha do Usuário.
- `cUserName (C, 15)` à Nome do Usuário sem brancos a direita.
- `cAcesso (C, 128)` à Permissões de Acesso do Usuário.
- `__UserID (C, 6)` à Número de identificação do usuário no cadastro de usuários. Ex.: "000006".
- `cNivel (N, 1)` à Nível de Acesso do Usuário podendo variar de 0 a 9 sendo que 9 indica um Administrador.
- `cModulo (C, 3)` à Sigla do Módulo que está em uso no momento. Ex.: "EST" para o Módulo de Estoque.
- `nModulo (N, 1)` à Número do Módulo que está em uso no momento. Ex.: 4 para o Módulo de Estoque.
- `aEmpresas (A)` à Array contendo tantos elementos quanto forem as empresas e filiais cadastradas no sistema sendo que cada elemento se encontra no formato "EEFF" (Empresa + Filial) Ex.: "5501".
- `cNumEmp (C, 4)` à Número da Empresa + Filial que está em uso no momento. Ex.: "9901".
- `cEmpAnt (C, 2)` à Número da Empresa que está em uso no momento. Ex.: "55".

- `cFilAnt (C, 2)` à Número da Filial que está em uso no momento. Ex.: "01".
- `cArqEmp (C, 12)` à Nome do arquivo de empresas com sua extensão. Ex.: "SIGAMAT.EMP".
- `cArqMnu (C, 12)` à Nome do arquivo de menu que está em uso no momento + a sua extensão. Ex.: "SIGAEST.MNU".
- `_NomeExec (C, 12)` à Nome do arquivo executável que está em uso no momento + a sua extensão. Ex.: "SIGAEST.EXE".
- `cFOpened (C)` à String com os Alias de todos os arquivos que estão abertos no momento.
- `cVersao (C)` à Versão do Sistema.
- `tInicio (C, 8)` à Horário em que foi iniciado o sistema no formato "HH:MM:SS".
- `__Language (C)` à Língua atual do Sistema. Ex.: "PORTUGUESE".
- `cPaisLoc (C, 3)` à Sigla do País para qual está configurado o Sistema. Ex.: "BRA".
- `__TTSinUse (L)` à Indica se o controle de transações está ativo no sistema. Retorna .T. se `MV_TTS = "S"` e .F. se for igual a "N".
- `__lSX8 (L)` à Usada em conjunto com as funções `GetSX8Num()` e `ConfirmSX8()`. Provavelmente para controlar se o número já está reservado para outro usuário.

Este Manual foi elaborado com base na Documentação Eletrônica Microsiga e outros materiais que foram disponibilizados do grupo de discussão Siga-br.

Esta documentação não se restringe apenas as funções de Programação abrange outros assuntos em menor escala.

O índice foi colaboração de um outro colega que não quis se identificar no momento.

Esta é a primeira versão oficial deste manual, caso algum colega tenha documentação e queira compartilhar é só nos enviar, pode também contribuir com sugestões para a melhoria desta documentação.

Autor :Waterloo Ferreira da Silva

waterloof@gmail.com

Site Interessantes sobre o assunto:

<http://www.microsiga.com.br/>

<http://www.clipx.net/ng/fivewin/index.php>

Este site contém help das funções do fivewin que em muitos casos são idênticas as utilizadas em advpl.

http://mateus.kicks-ass.net/advpl/index.php/Guia_de_Funções

<http://www.helpfacil.com.br/>