

Todos os direitos autorais reservados pela **TOTVS S.A.**

Proibida a reprodução total ou parcial, bem como a armazenagem em sistema de recuperação e a transmissão, de qualquer modo ou por qualquer outro meio, seja este eletrônico, mecânico, de fotocópia, de gravação, ou outros, sem prévia autorização por escrito da proprietária.

O desrespeito a essa proibição configura em apropriação indevida dos direitos autorais e patrimoniais da TOTVS.

Conforme artigos 122 e 130 da LEI no. 5.988 de 14 de Dezembro de 1973.

Programação ADVPL Avançado

Protheus – Versão 12



Sumário

1. Objetivo.....	4
2. Programação de Atualização	4
2.1. Modelo1() ou AxCadastro().....	5
2.2. Mbrowse().....	6
2.2.5. BrwLegenda ()	14
2.3. MarkBrowse()	16
2.4. Enchoice	20
2.5. EnchoiceBar	23
2.6. Modelo2().....	24
2.7. Modelo3().....	26
3. Transações.....	37
3.1. Begin Transaction.....	38
3.2. BeginTran	38
4. Relatórios.....	39
4.1. TReport.....	39
4.2. Funções Utilizadas para Desenvolvimento de Relatórios	50
4.3. AjustaSX1().....	51
5. Utilizando Querys no Protheus	57
5.1. Embedded SQL	61
6. Manipulação de arquivos.....	62
6.1. Geração e leitura de arquivos em formato texto	62
6.2. 1ª Família de funções de gravação e leitura de arquivos texto.....	62
6.3. 2ª Família de funções de gravação e leitura de arquivos texto.....	71
7. Captura de múltiplas informações (Multi-Lines).....	76
7.2. TCBrowse	84
8. FWTemporaryTable	87
9. Arquitetura MVC	89
9.1. Principais funções da aplicação em AdvPL utilizando o MVC	90
9.2. O que é a função ModelDef?	90
9.3. O que é a função ViewDef?	91
9.4. O que é a função MenuDef?	91
10. Novo comportamento na interface	92
10.1. Aplicações com Browsers (FWMBrowse)	92
10.2. Exemplo completo de Browse	95
11. Construção de aplicação ADVPL utilizando MVC.....	95
12. Criando o MenuDef.....	95
13. Construção da função ModelDef	96

13.1. Construção de uma estrutura de dados (FWFormStruct)	97
13.2. Criação de componente no modelo de dados (AddFields)	98
13.3. Criação de um componente na interface (AddField).....	98
13.4. Descrição dos componentes do modelo de dados (SetDescription)	99
13.5. Finalização de ModelDef	99
14. Construção da função ViewDef	100
14.1. Exibição dos dados (CreateHorizontalBox / CreateVerticalBox).....	100
14.2. Relacionando o componente da interface (SetOwnerView).....	101
14.3. Finalização da ViewDef	101
15. Objetos de interface.....	102
15.1. Réguas de processamento	104
15.2. ListBox().....	114
15.3. ScrollBox()	119
MÓDULO 06: ADVPL Orientado à objetos	122
16. Componentes da interface visual do ADVPL	122
16.1. Particularidades dos componentes visuais	129
17. Aplicações com a interface visual do ADVPL	131
17.1. Imagens pré-definidas para as barras de botões.....	131
APÊNDICES	133
BOAS PRÁTICAS DE PROGRAMAÇÃO	133
18. Arredondamento	133
19. Utilização de Identação.....	133
20. Capitulação de Palavras-Chave	134
20.1. Palavras em maiúsculo.....	134
21. Técnicas de programação eficiente	135

1. Objetivo

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

a) Conceitos a serem aprendidos

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

b) Habilidades e técnicas a serem aprendidas

- Desenvolvimento de aplicações voltadas ao ERP Protheus
- Análise de fontes de média complexidade
- Desenvolvimento de aplicações básicas com orientação a objetos

c) Atitudes a serem desenvolvidas

- Adquirir conhecimentos através da análise das funcionalidades disponíveis no ERP Protheus;
- Estudar a implementação de fontes com estruturas orientadas a objetos em ADVPL;
- Embasar a realização de outros cursos relativos a linguagem ADVPL

2. Programação de Atualização

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

Basicamente são três os modelos mais utilizados:

- **Modelo 1 ou AxCadastro:** Para cadastramentos em tela cheia. Exemplo: Cadastro de Cliente.
- **Modelo 2:** Cadastramentos envolvendo apenas uma tabela, mas com um cabeçalho e, opcionalmente, um rodapé e um corpo com quantidade ilimitada de linhas. Ideal para casos em que há dados que se repetem por vários itens e que, por isso, são colocados no cabeçalho. Exemplo: Pedido de Compra.
- **Modelo 3:** Cadastramentos envolvendo duas tabelas, um com dados de cabeçalho e outro digitado em linhas com os itens. Exemplo: Pedido de Vendas, Orçamento etc.

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

2.1. Modelo1() ou AxCadastro()

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

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

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

Parâmetros:

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

Exemplo: Função AxCadastro()

```
#include "protheus.ch"

User Function XCadSA2()
  Local cAlias := "SA2"
  Local cTitulo := "Cadastro de Fornecedores"
  Local cVldExc := ".T."
  Local cVldAlt := ".T."
  dbSelectArea(cAlias)
  dbSetOrder(1)
  AxCadastro(cAlias, cTitulo, cVldExc, cVldAlt)
Return Nil
```

Exemplo: Função de validação da alteração

```
User Function VldAlt(cAlias, nReg, nOpc)
  Local lRet := .T.
  Local aArea := GetArea()
  Local nOpcao := 0

  nOpcao := AxAlterar(cAlias, nReg, nOpc)

  If nOpcao == 1
    MsgInfo("Alteração concluída com sucesso!")
  Endif
  RestArea(aArea)
Return lRet
```

Exemplo: Função de validação da exclusão

```
User Function VldExc(cAlias,nReg,nOpc)

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

nOpcao := AxExclui(cAlias,nReg,nOpc)
If nOpcao == 1
    MsgInfo("Exclusão concluída com sucesso!")
Endif

RestArea(aArea)
Return lRet
```

2.2. Mbrowse()

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

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

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

Parâmetros:

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

Parâmetros:

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

Variáveis private adicionais

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

Informações passadas para funções do aRotina:

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

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

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

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

Exemplo: Função Mbrowse()

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

```
User Function MBrwSA1()

Local cAlias      := "SA1"
Private cCadastro := "Cadastro de Clientes"

Private aRotina := {}

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

dbSelectArea(cAlias)
dbSetOrder(1)
mBrowse( , , , , cAlias)

Return Nil
```

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

```
#include "protheus.ch"

User Function MBrwSA1()

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

AADD(aRotina, {"Pesquisar" , "AxPesqui" , 0, 1})
AADD(aRotina, {"Visualizar" , "AxVisual" , 0, 2})
AADD(aRotina, {"Incluir" , "U_Inclui" , 0, 3})
AADD(aRotina, {"Alterar" , "AxAlterar" , 0, 4})
AADD(aRotina, {"Excluir" , "AxDeleta" , 0, 5})

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

Return Nil
```

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

```
User Function Inclui(cAlias, nReg, nOpc)
```

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

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

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

Return Nil
```

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

```
#include "protheus.ch"

User Function Exclui(cAlias, nReg, nOpc)

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

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

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

Return Nil
```

2.2.1. AxFunctions()

Conforme mencionado nos tópicos sobre as interfaces padrões AxCadastro() e Mbrowse(), existem funções padrões da aplicação ERP que permitem a visualização, inclusão, alteração e exclusão de dados em formato simples. Estas funções são padrões na definição da interface AxCadastro() e podem ser utilizadas também da construção no array aRotina utilizado pela Mbrowse(), as quais estão listadas a seguir:

- ☐ **AXPESQUI()**
- ☐ **AXVISUAL()**
- ☐ **AXINCLUI()**
- ☐ **AXALTERA()**
- ☐ **AXDELETA()**

AXPESQUI()

Sintaxe

AXPESQUI()

Descrição

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

AXVISUAL()

Sintaxe

AXVISUAL(cAlias, nReg, nOpc, aAcho, nColMens, cMensagem, cFunc,; aButtons, IMaximized)

Descrição

Função de visualização padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

AXINCLUI()

Sintaxe

AxInclui(cAlias, nReg, nOpc, aAcho, cFunc, aCpos, cTudoOk, IF3,; cTransact, aButtons, aParam, aAuto, IVirtual, IMaximized)

Descrição

Função de inclusão padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

AXALTERA()

Sintaxe

AxAlterar(cAlias, nReg, nOpc, aAcho, cFunc, aCpos, cTudoOk, IF3,; cTransact, aButtons, aParam, aAuto, IVirtual, IMaximized)

Descrição

Função de alteração padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

AXDELETA()

Sintaxe

AXDELETA(cAlias, nReg, nOpc, cTransact, aCpos, aButtons, aParam,; aAuto, IMaximized)

Descrição

Função de exclusão padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

Exercício

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

2.2.2. FilBrowse()

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

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

Parâmetros:

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

2.2.3. EndFilBrw()

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

Sintaxe: EndFilBrw(cAlias, aQuery)

Parâmetros:

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

2.2.4. PesqBrw()

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

Sintaxe: PesqBrw(cAlias , nReg, bBrwFilter)

Parâmetros:

cAlias	Alias ativo definido para a Mbrowse()
nReg	Número do registro

bBrwFilter

Bloco de Código que contém a FilBrowse()

Ex: bBrwFilter := { || FilBrowse(cAlias, aQuery, cFiltro, IShowProc) }

2.2.5. BrwLegenda ()

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

Sintaxe: BrwLegenda(cCadastro , cTitulo, aLegenda)

Parâmetros:

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

Lista de cores disponíveis no Protheus

- ☐ BR_AMARELO
- ☐ BR_AZUL
- ☐ BR_BRANCO
- ☐ BR_CINZA
- ☐ BR_LARANJA
- ☐ BR_MARRON
- ☐ BR_VERDE
- ☐ BR_VERMELHO
- ☐ BR_PINK
- ☐ BR_PRETO

Exemplo: Mbrowse() utilizando as funções acessórias

```
#Include "Protheus.ch"

User Function MBrwSA2()

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

```

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

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

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

dbSelectArea(cAlias)
dbSetOrder(1)

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

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

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

Return Nil

//+-----
//| Função: BInclui - Rotina de Inclusão
//+-----

User Function BInclui(cAlias,nReg,nOpc)
Local nOpcao := 0
nOpcao := AxInclui(cAlias,nReg,nOpc)

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

Return Nil

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

```

```

Local nOpcao := 0

nOpcao := AxAltera(cAlias,nReg,nOpc)

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

Return Nil

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

Local nOpcao := 0

nOpcao := AxDeleta(cAlias,nReg,nOpc)

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

Return Nil

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

Local ALegenda ={}

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

BrwLegenda(cCadastro, "Legenda", aLegenda)

Return Nil
    
```

2.3. MarkBrowse()

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

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

Parâmetros:

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

Informações passadas para funções do aRotina:

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

2.3.1. Funções de Apoio

- **GetMark:** define a marca atual.
- **IsMark:** avalia se um determinado conteúdo é igual a marca atual.
- **ThisMark:** captura a marca em uso.
- **ThisInv:** indica se foi usado o recurso de selecionar todos (inversão).

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

Exemplo: Função MarkBrow() e acessórios

```
#include "protheus.ch"

USER FUNCTION MkBrwSA1()

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

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

AADD(aRotina, {"Visualizar Lote", "U_VisLote", 0, 5})

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

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

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

Return Nil
```

Exemplo: Função VisLote() – utilização das funções acessórios da MarkBrow()

```
USER FUNCTION VisLote()

Local cMarca     := ThisMark()
Local nX := 0
Local lInvert    := ThisInv()
Local cTexto     := ""
Local cEOL       := CHR(10)+CHR(13)
Local oDlg
Local oMemo
```

```

DbSelectArea("SA1")
DbGoTop()

While SA1->(!EOF())

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

    SA1->(dbSkip())

Enddo

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

    For nX := 1 to Len(aRecSel)

        cTexto+=aRecSel[nX][2]+Space(1)+"|"+Space(2)+aRecSel[nX][3] ;
            + Space(3)+"|"

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

    Next nX

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

RETURN

```

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

```

STATIC FUNCTION LimpaMarca()
Local nX := 0
For nX := 1 to Len(aRecSel)
    SA1->(DbGoto(aRecSel[nX][1]))
    RecLock("SA1",.F.)
    SA1->A1_OK := SPACE(2)
    MsUnLock()
Next nX
Return()

```

2.4. Enchoice

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

Sintaxe: Enchoice (cAlias [nReg] nOpc [aCRA] [cLetras] [cTexto] [aAcho] [aPos] [aCpos] [nModelo] [nColMens] [cMensagem] [cTudoOk] [oWnd] [IF3] [IMemoria] [IColumn] [caTela] [INoFolder] [IProperty] [aField] [aFolder] [ICreate] [INoMDIStrech] [cTela])

Nome	Tipo	Descrição	Obrigatório
cAlias	Caracter	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada	X
nReg	Nulo	Parâmetro não utilizado	
nOpc	Numérico	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização)	X
aCRA	Nulo	Parâmetro não utilizado	
cLetras	Nulo	Parâmetro não utilizado	
cTexto	Nulo	Parâmetro não utilizado	
aAcho	Vetor	Vetor com nome dos campos que serão exibidos. Os campos de usuário sempre serão exibidos se não existir no parametro um elemento com a expressão "NOUSER"	
após	Vetor	Vetor com coordenadas para criação da enchoice no formato { , , }	
aCpos	Vetor	Vetor com nome dos campos que poderão ser editados	
nModelo	Numérico	Se for diferente de 1 desabilita execução de gatilhos estrangeiros	
nColMens	Nulo	Parâmetro não utilizado	
cMensagem	Nulo	Parâmetro não utilizado	
cTudoOk	Nulo	Parâmetro não utilizado	
oWnd	Objeto	Objeto (janela, painel, etc) onde a enchoice será criada	
IF3	Lógico	Indica se a enchoice esta sendo criada em uma consulta F3 para utilizar variáveis de memória	
IMemoria	Lógico	Indica se a enchoice utilizará variáveis de memória ou os campos da tabela na edição	
IColumn	Lógico	Indica se a apresentação dos campos será em forma de coluna	
caTela	Caracter	Nome da variável tipo "private" que a enchoice utilizará no lugar da variável aTela	
INoFolder	Lógico	Indica se a enchoice não irá utilizar as Pastas de Cadastro (SXA)	

IProperty	Lógico	Indica se a enchoice não utilizará as variáveis tipo "private" aTela e aGets, somente suas propriedades com seus respectivos nomes	
aField	Vetor	Vetor com os campos que serão mostrados na Enchoice caso o SX3 não seja utilizado	
aFolder	Vetor	Vetor com o nome das pastas caso o SX3 não seja utilizado	
ICreate	Lógico	Indica se cria as pastas especificadas no parâmetro aFolder	
INoMDIStrech	Lógico	Define se o objeto não será alinhado conforme o espaço existente na janela	
cTela	Caracter	Campo reservado	

```

Local aSize      := {}
Local aPObj      := {}

// Retorna a area util das janelas Protheus
aSize := MsAdvSize()

// Será utilizado três áreas na janela
// 1ª - Enchoice, sendo 80 pontos pixel

AADD( aObj, { 100, 080, .T., .F. })
AADD( aObj, { 100, 100, .T., .T. })
AADD( aObj, { 100, 015, .T., .F. })

// Cálculo automático da dimensões dos objetos (altura/largura) em pixel

aInfo := { aSize[1], aSize[2], aSize[3], aSize[4], 3, 3 }
aPObj := MsObjSize( aInfo, aObj )

// Cálculo automático de dimensões dos objetos MSGET
oDialog:=MSDialog():New(aSize[7],aSize[1],aSize[6],aSize[5],,
    "MSDialog",,,,,CLR_BLACK,CLR_WHITE,,.T.)

EnChoice( cAlias1, nReg, nOpc, , , , , aPObj[1])

oDialog:Activate(,,,.T.)

```

Vale lembrar que nós programadores reaproveitamos muito o que já existe, isto é para simplesmente ganharmos tempo, e no caso da utilização da função Enchoice é preciso criar as variáveis de memórias que levam o mesmo nome dos campos da tabela em questão. Por exemplo o campo A2_NOME da tabela SA2 (cadastro de fornecedores) quando queremos referenciar o campo usa-se o prefixo da tabela e o campo em questão, desta forma:

- SA2->A2_NOME

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

- M->A2_NOME

E para criar variáveis com o nome do campo utilizamos a função:

RegToMemory()

Sintaxe	RegToMemory(cAlias, lInc , lDic , lInitPad , cStack)
Descrição	<p>Esta função inicializa as variáveis de memória utilizadas pela interfaces.</p> <p>cAlias: Alias da tabela que terá suas variáveis inicializadas.</p> <p>lInc: Indica se a inicialização será baseada numa operações de inclusão (.T.) ou manutenção (.F.). A diferença entre elas é que na operação de inclusão os valores são inicializados vazios e na manutenção com os dados do registro posicionado.</p> <p>lDic: Indica se a inicialização das variáveis será baseada no dicionário de dados ou apenas nos dados da WorkArea aberta. A diferença entre elas são os campos virtuais que somente são inicializados com a opção .T. - Dicionário de dados.</p> <p>lInitPad: Indica se o inicializador padrão do dicionário de dados será executado. Este parametro somente será acionado se o parametro Expl3 for configurado como .T. - Dicionário de Dados.</p> <p>cStack: Qualquer parâmetro reservado</p>

As informações contidas no SX3 definirão se o campo deverá ser exibido, editado e qual tipo de objeto será utilizado (Get, Combo, Memo, CheckBox).

Para facilitar as validações possui funções para ajudar nas validações na Enchoice, para verificar se todos os campos foram digitados podemos utilizar a função:

Obrigatorio()

Sintaxe	obrigatório (aGets, aTela, uPar3, lShow)
Descrição	<p>aGets: Variável PRIVATE tratada pelo objeto Enchoice(), previamente definida no fonte.</p> <p>aTela: Variável PRIVATE tratada pelo objeto Enchoice(), previamente definida no fonte.</p> <p>uPar3: Não utilizado.</p> <p>lShow: Lógico determina se exibirá o help caso algum campo obrigatório não tenha sido preenchido. Default é .T.</p> <p>Função retorna um valor lógico .T. indicando se todos os campos obrigatórios foram preenchidos</p>

FieldPos()

Sintaxe	FieldPos(X3_CAMPO)
Descrição	A função FieldPos tem a funcionalidade de retornar a posição de um determinado campo dentro da estrutura do alias corrente. Caso o campo não exista na estrutura, é retornado zero.

FieldGet()

Sintaxe	FieldGet(<nFieldPos>)
Descrição	nFieldPos: Número da posição ordinal do campo na tabela SX3

FieldName ()

Sintaxe	FieldName (< cNomeCampo >)
Descrição	cNomeCampo: Retorna uma string contendo o nome do campo especificado. Se o parâmetro (nPosição) for maior que o total de campos do alias atual ou não tenha um alias aberto na área de trabalho, a função retornará uma string vazia ("").

FieldPut ()

Sintaxe	FieldPut (<nCampo>, <Conteúdo>)
Descrição	É uma função de banco de dados que atribui o valor de uma expressão a um campo da tabela atual, informado a partir da posição ordinal deste na estrutura da tabela. Através dela, podemos atribuir um valor a um campo utilizando a sua posição na estrutura ao invés do nome do campo. Isto facilita a criação de funções genéricas para serviços de manipulação de dados, entre outras aplicações, permite setar um conteúdo a um campo sem a utilização do operador de macro-execução.

2.5. EnchoiceBar

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

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

Parâmetros:

oDlg	Objeto	Janela onde a barra será criada.
bOk	Objeto	Bloco de código executado quando clicado botão Ok.
bCancelar	Objeto	Bloco de código executado quando clicado.
IMensApag	Lógico	Indica se ao clicar no botão Ok aparecerá uma tela de confirmação de exclusão. Valor padrão falso.

Abotoes	Vetor	Vetor com informações para criação de botões adicionais na barra. Seu formato é {bitmap, bloco de código, mensagem}. AADD(aButtons,{"CLIPS",{" AllwaysTrue()","Usuário")})
----------------	-------	--

2.6. Modelo2()

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

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

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

Parâmetros:

cTitulo	Título da janela
aCab	Array contendo as informações que serão exibidas no cabeçalho na forma de Enchoice() aCab[n][1] (Caractere) := Nome da variável private que será vinculada ao campo da Enchoice(). aCab[n][2] (Array) := Array com as coordenadas do campo na tela {Linha, Coluna} aCab[n][3] (Caractere) := Título do campo na tela aCab[n][4] (Caractere) := Picture de formatação do get() do campo. aCab[n][5] (Caractere) := Função de validação do get() do campo. aCab[n][6] (Caractere) := Nome da consulta padrão que será executada para o campo via tecla F3 aCab[n][7] (Lógico) := Se o campo estará livre para digitação.
aRoda	Array contendo as informações que serão exibidas no cabeçalho na forma de Enchoice(), no mesmo formato que o aCab.
aGrid	Array contendo as coordenadas da GetDados() na tela. Padrão := {44,5,118,315}
nOpc	Opção selecionada na MBrowse, ou que deseje ser passada para controle da Modelo2, aonde: 2 – Visualizar 3 - Incluir 4 - Alterar 5 - Excluir
cLinhaOk	Função para validação da linha na GetDados()
cTudoOk	Função para validação na confirmação da tela de interface da Modelo2().

Retorno:

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

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


```
#include "protheus.ch"

USER FUNCTION MBrw2Sx5()

Local cAlias := "SX5"

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

AADD(aRotina,{"Pesquisar"          ,"AxPesqui"  ,0,1})
AADD(aRotina,{"Visualizar"        ,"U_SX52Vis" ,0,2})
AADD(aRotina,{"Incluir"           ,"U_SX52Inc" ,0,3})
AADD(aRotina,{"Alterar"           ,"U_SX52Alt" ,0,4})
AADD(aRotina,{"Excluir"           ,"U_SX52Exc" ,0,5})

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

Return
//-----
USER FUNCTION SX52INC(cAlias,nReg,nOpc)

Local cTitulo:= "Inclusao de itens - Arquivo de Tabelas"
Local aCab      := {} // Array com descricao dos campos do Cabecalho
Local aRoda     := {} // Array com descricao dos campos do Rodape
Local aGrid := {80,005,050,300} //Array com coordenadas da
//GetDados no modelo2 - Padrao: {44,5,118,315} Linha Inicial -
//Coluna Inicial - +Qts Linhas - +Qts Colunas : {080,005,050,300}

Local cLinhaOk := "AlwaysTrue()" // Validacoes na linha
Local cTudoOk  := "AlwaysTrue()" // Validacao geral da GetDados
Local lRetMod2 := .F. // Retorno da função Modelo2 - .T. Confirmou
// .F. Cancelou
Local nColuna  := 0

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

// Variaveis para campos da Enchoice()
Private cX5Filial := xFilial("SX5")
Private cX5Tabela := SPACE(5)
// Montagem do array de cabeçalho
//AADD(aCab,{"Variável",{L,C}"Título","Picture","Valid","F3",lEnable})
AADD(aCab,{"cX5Filial"      ,{015,010} ,"Filial","@!",,,.F.})
AADD(aCab,{"cX5Tabela"     ,{015,080} ,"Tabela","@!",,,.T.})

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

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

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

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

Next nColuna

aCols[1][Len(aHeader)+1] := .F. // Linha não deletada
lRetMod2 := Modelo2(cTitulo,aCab,aRoda,aGrid,nOpc,cLinhaOk,cTudoOk)
IF lRetMod2
    //MsgInfo("Você confirmou a operação","MBRW2SX5")

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

2.7. Modelo3()

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

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

um pedido de vendas, onde terá um registro em uma tabela referente à cabeça do pedido e outra tabela com os registros referentes aos itens deste pedido de vendas.

Para ganharmos tempo não será apresentado aqui toda a explicação e montagens para a função EnchoiceBar, comando MsDialog, Say e MsGet e para os vetores aHeader e aCOLS, entretanto todos estes estarão na codificação do código fonte.

A tela de modelo 3 é constituído de MsDialog, EnchoiceBar, Enchoice, MsGetDados, Say e Get. é ele a função Enchoice, o que é?

2.7.1. Estrutura de um programa utilizando a Modelo3()

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

Estrutura do programa

Rotina principal

#Include "protheus.ch"

```
on xModelo3()
Private cCadastro := "Protótipo Modelo 3"
Private aRotina := {}
Private oCliente
Private oTotal
```

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

```

Private cCliente := ""
Private nTotal := 0

Private bCampo := {|nField| FieldName(nField) }

Private aSize := {}
Private aInfo := {}
Private aObj := {}
Private aPObj := {}
Private aPGet := {}

// Retorna a área útil das janelas Protheus
aSize := MsAdvSize()

// Será utilizado três áreas na janela
// 1ª - Enchoice, sendo 80 pontos pixel
// 2ª - MsGetDados, o que sobrar em pontos pixel é para este objeto
// 3ª - Rodapé que é a própria janela, sendo 15 pontos pixel
AADD( aObj, { 100, 080, .T., .F. })
AADD( aObj, { 100, 100, .T., .T. })
AADD( aObj, { 100, 015, .T., .F. })

// Cálculo automático da dimensões dos objetos (altura/largura) em pixel
aInfo := { aSize[1], aSize[2], aSize[3], aSize[4], 3, 3 }
aPObj := MsObjSize( aInfo, aObj )

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

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

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

Função de Inclusão

```

User Function Mod3Inc( cAlias, nReg, nOpc )
Local oDlg
Local oGet
Local nX := 0
Local nOpcA := 0

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

dbSelectArea( cAlias )
dbSetOrder(1)

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

```

Next nX

Mod3aHeader()
Mod3aCOLS( nOpc )

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

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

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

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

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

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

```

Função de Visualização, Alteração e Exclusão

```

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

dbSelectArea( cAlias )
dbSetOrder(1)

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

```

```

Next nX

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

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

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

    U_Mod3Cli()

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

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

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

Função para montar o vetor aHeader

```

Static Function Mod3aHeader()
Local aArea := GetArea()

dbSelectArea("SX3")
dbSetOrder(1)
dbSeek("ZA2")
While !EOF() .And. X3_ARQUIVO == "ZA2"
    If X3Uso(X3_USADO) .And. cNivel >= X3_NIVEL
        AADD( aHeader, { Trim( X3Titulo() ),;
            X3_CAMPO,;
            X3_PICTURE,;
            X3_TAMANHO,;
            X3_DECIMAL,;
            X3_DECIMAL,;
            X3_VALID,;
            X3_USADO,;
    
```

```

X3_TIPO,,
X3_ARQUIVO,,
X3_ARQUIVO,,
X3_ARQUIVO,,
X3_ARQUIVO,,
X3_ARQUIVO,,
X3_CONTEXT})

Endif
dbSkip()

End
RestArea(aArea)
Return

```

Função para montar o vetor aCols

```

Static Function Mod3aCOLS( nOpc )
Local aArea := GetArea()
Local cChave := ""
Local cAlias := "ZA2"
Local nI := 0

If nOpc <> 3
    cChave := ZA1->ZA1_NUM
    dbSelectArea( cAlias )
    dbSetOrder(1)
    dbSeek( xFilial( cAlias ) + cChave )

While !EOF() .And. ZA2->( ZA2_FILIAL + ZA2_NUM ) == xFilial(cAlias)+ cChave
    AADD( aREG, ZA2->( RecNo() ) )
    AADD( aCOLS, Array( Len( aHeader ) + 1 ) )
    For nI := 1 To Len( aHeader )
        If aHeader[nI,10] == "V"
            aCOLS[Len(aCOLS),nI] := CriaVar(aHeader[nI,2],.T.)
        Else
            aCOLS[Len(aCOLS),nI] := FieldGet(FieldPos(aHeader[nI,2]))
        Endif
    Next nI
    aCOLS[Len(aCOLS),Len(aHeader)+1] := .F.
    dbSkip()
EndDo

Else
    AADD( aCOLS, Array( Len( aHeader ) + 1 ) )
    For nI := 1 To Len( aHeader )
        aCOLS[1, nI] := CriaVar( aHeader[nI, 2], .T. )
    Next nI
    aCOLS[1, GdFieldPos("ZA2_ITEM")] := "01"
    aCOLS[1, Len( aHeader ) + 1 ] := .F.
Endif

Restarea( aArea )
Return

```

Função para atribuir o nome do cliente a variável

```
User Function Mod3Cli()
```

```
cCliente := Posicione("SA1",1,xFilial("SA1")+ M->(ZA1_CLIENT+ZA1_LOJA),;
    "A1_NREDUZ" )
oCliente:Refresh()
Return (.T.)
```

Função para validar a mudança de linha na MsGetDados()

```
User Function Mod3LOk()
Local nI := 0

nTotal := 0
For nI := 1 To Len( aCOLS )
    If aCOLS[nI,Len(aHeader)+1]
        Loop
        Endif
        nTotal+=Round(aCOLS[nI,GdFieldPos("ZA2_QTDVEN")]*;
            aCOLS[nI,GdFieldPos("ZA2_PRCVEN")],2)
Next nI
oTotal:Refresh()

Return(.T.)

Static Function Mod3TOk()
Local nI := 0
Local lRet := .T.

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

Return( lRet )
```

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

```
Static Function Mod3Grv( nOpc, aAlterar)
Local nX := 0
Local nI := 0
```



```
// Se for inclusão
If nOpc == 3
  // Grava os itens
  dbSelectArea("ZA2")
  dbSetOrder(1)
  For nX := 1 To Len( aCOLS )
    If !aCOLS[ nX, Len( aCOLS ) + 1 ]
      RecLock( "ZA2", .T. )
      For nI := 1 To Len( aHeader )
        FieldPut( FieldPos( Trim( aHeader[nI, 2] ) ), aCOLS[nX,nI]
      )

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

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

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

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

```

        Endif
        MsUnLock()
Next nX

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

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

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

2.7.2. Função Modelo3()

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

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

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

A função Modelo3() não implementa as regras de visualização, inclusão, alteração e exclusão, como uma AxCadastro() ou AxFunction().

A inicialização dos campos utilizados na Enchoice() deve ser realizadas pela rotina que “suporta” a execução da Modelo3(), normalmente através do uso da função RegToMemory().

Da mesma forma, o Browse deve ser tratado por esta rotina, sendo comum a Modelo3() estar vinculada ao uso de uma MBrowse().

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

Parâmetros:

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

Retorn:

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

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

```
#INCLUDE "protheus.ch"

User Function MbrwMod3()

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

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

dbSelectArea(cAlias)
```


Este documento é de propriedade da TOTVS. Todos os direitos reservados. ©

3. Transações

- BEGIN TRANSACTION / END TRANSACTION
- Begin / DisarmTransaction / EndTran

3.1. Begin Transaction

Transações são unidades lógicas que ocorrem em operações de banco de dados e são responsáveis por manter a consistência, isolamento, coerência e durabilidade dos dados. Sem as transações corre-se o risco de termos dados inconsistentes o que tornaria duvidoso ou questionável qualquer operação de banco de dados.

Este comando define que as operações seguintes, delimitadas pelo comando END TRANSACTION, devem ser processadas como uma transação, ou seja, como um bloco único e indivisível. Durante uma recuperação de falha, todas as operações de manipulação de dados realizadas serão integralmente desfeitas, além de prover isolamento entre acessos concorrentes na mesma massa de dados.

Exemplo:

```
BEGIN TRANSACTION
  RecLock ("SB2")
  ...
  MsUnlock ()
END TRANSACTION
```

3.2. BeginTran

Inicializa uma transação que permite o término da transação por meio de uma condição.

O comando END TRANSACTION não pode ter sua interpretação vinculada a uma condição. Nestes casos ocorrerá um erro de compilação indicando que o bloco aberto pelo comando BEGIN TRANSACTION não foi corretamente fechado.

No uso do controle de transação através do bloco BeginTran() ... EndTran() é recomendável a utilização da função MsUnlockAll() para destravar todos os registros que estejam eventualmente locados.

Exemplo:

```
AADD(xAutoCab, {"A1_FILIAL", xFilial("SA1") , Nil})
AADD(xAutoCab, {"A1_COD" , "000001" , Nil})
AADD(xAutoCab, {"A1_LOJA" , "01" , Nil})
AADD(xAutoCab, {"A1_NOME" , "TESTE-000001" , Nil})

BeginTran()
lMsErroAuto := .F.
MsExecAuto({|x,y| MATA030(x,y)}, xAutoCab, 3)

IF lMsErroAuto
  DisarmTransaction()
ELSE
  EndTran()
ENDIF
```

```
MsUnlockAll()
```

4. Relatórios

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

4.1. TReport

O Protheus oferece o recurso personalização para alguns relatórios de cadastros e movimentações do sistema. Ele tem como principais funcionalidades a definição de cores, estilos, tamanho, fontes, quebras, máscara das células para cada seção, criação de fórmulas e funções (Soma, Média, etc.), possibilidade de salvar as configurações por usuário e criação de gráficos.

Com a funcionalidade de Relatórios Personalizáveis, o usuário pode modificar os relatórios padrões, criando seu próprio layout.

relat:comissao

SIGA:MATR424/MPR.11

Hora: 18:34:18

Relatório de Comissões

Folha: 1

Dt.Ref: 06/10/98

Emissão: 06/10/98

Vendedor	Nome	Parâmetro	Cliente	Nome	DT Comissao	Vencido Original	DT Base	Data Pagto	Percento	VB Total	VB Base	%
000001	KNO TESTEIRA											
81330		000001	SODOPHAB		05/10/98	05/10/98	//	10/10/98		2500.00	500.00	1
81330		000001	TECCHI		05/10/98	10/10/98	06/10/98	10/10/98		1700.00	300.00	1
81330		000001	TECCHI		05/10/98	10/10/98	06/10/98	10/10/98		1700.00	1400.00	1
81331		000001	SODOPHAB		05/10/98	10/10/98	//	10/10/98		4000.00	800.00	1
Total das Comissões por vendedor										13.000.00	3.000.00	
000001	KNO TESTEIRA											
Total de		Total (C) R\$										
30.00		277.20										
Vendedor	Nome											
000001	CELNO RABOZUEIRA											
81332		000001	TECCHI		05/10/98	10/10/98	06/10/98	10/10/98		1.000.00	700.00	1
81332		000001	TECCHI		05/10/98	10/10/98	06/10/98	10/10/98		1.000.00	2.000.00	1
81330		000001	SODOPHAB		05/10/98	10/10/98	//	10/10/98		3.000.00	1.000.00	1
81331		000001	SODOPHAB		05/10/98	10/10/98	//	10/10/98		1.700.00	100.00	1
Total das Comissões por vendedor										13.700.00	4.400.00	
000001	CELNO RABOZUEIRA											
Total de		Total (C) R\$										
37.00		826.80										
Total Geral										26.700.00	7.400.00	

Vale lembrar que nem todos os relatórios são personalizáveis. Por exemplo, relatórios que tenham layout pré-definidos por lei e formulários (boletos, notas-fiscais, etc) não poderão ser alterados.

Os relatórios personalizados são gravados com extensão .PRT, diferenciando-se dos relatórios padrões que recebem a extensão .###R.

Descrição

O TReport é uma classe de impressão que substitui as funções SetPrint, SetDefault, RptStatus e Cabec.

A classe TReport permite que o usuário personalize as informações que serão apresentadas no relatório, alterando fonte (tipo, tamanho, etc), cor, tipo de linhas, cabeçalho, rodapé, etc.

Estrutura do componente TReport:

- O relatório (TReport) contém 1 ou mais seções (TRSection);
- Uma seção (TRSection) pode conter 1 ou mais seções;
- A seção (TRSection) contém células pré-definidas e células selecionadas pelo usuário;
- A seção (TRSection) também contém as quebras (TRBreak) para impressão de totalizadores (TRFunction);
- Os totalizadores são incluídos pela seção que automaticamente inclui no relatório (TReport).
- Pré-Requisitos

Para utilizar o TReport, verifique se o seu repositório está com o Release 4 do Protheus-8, ou versão superior.

A função `TRepInUse()` verifica se a lib do `TReport` está liberada no repositório em uso. O retorno é uma variável lógica.

```
#include "protheus.ch"

User Function MyReport()
Local oReport

If TRepInUse() //verifica se a opção relatórios personalizáveis está disponível
    Pergunte("MTR025",.F.)

    oReport := ReportDef()
    oReport:PrintDialog()
EndIf
Return
```

Relatório Simples com uma Section:

```
User Function RSimples()
Local oReport := nil
//Chama a função para carregar a Classe tReport
oReport := RptDef()
oReport:PrintDialog()
Return()

Static Function RptDef()

Local oReport := Nil
Local oSection1:= Nil
Local oBreak
Local oFunction

//Sintaxe:
//Classe TReport
//cNome: Nome físico do relatório
//cTitulo: Titulo do Relario
//cPergunta: Nome do grupo de pergunta que sera carredo em parâmetros
//bBlocoCodigo: Execura a função que ira alimenter as TRSection
//TReport():New(cNome,cTitulo,cPerguntas,bBlocoCodigo,cDescricao)

oReport:=TReport():New("Exemplo01", "Cadastro Produtos",/* cPergunta \*,;
{|oReport| ReportPrint( oReport ) }, "Descrição do meu relatório")

// Relatorio em retrato
oReport:SetPortrait()

// Define se os totalizadores serão impressos em linha ou coluna
oReport:SetTotalInLine(.F.)

//Monstando a primeira seção
oSection1:= TRSection():New(oReport, "Produtos", {"SB1"}, NIL, .F., .T.)
TRCell():New(oSection1, "B1_COD", "SB1","Produto", "@!",30 )
TRCell():New(oSection1, "B1_DESC", "SB1","Descrição", "@!",100)
TRCell():New(oSection1, "B1_LOCPAD", "SB1","Arm.Padrao", "@!",20 )
TRCell():New(oSection1, "B1_POSIPI", "SB1","NCM", "@!",30 )
```

```
//O parâmetro que indica qual célula o totalizador se refere ,
//será utilizado para posicionamento de impressão do totalizador quando
//estiver definido que a impressão será por coluna e como conteúdo para a
//função definida caso não seja informada uma fórmula para o totalizador
TRFunction():New(oSection1:Cell("B1_COD"),NIL,"COUNT",,,,,.F.,.T.)

Return(oReport)

Static Function ReportPrint(oReport)
    Local oSection1 := oReport:Section(1)
    Local cQuery     := ""
    Local cNcm       := ""
    Local lPrim      := .T.
    Local cAlias      := GetNextAlias()
    cPart := "% AND B1_COD >= '" + MV_PAR01 + "' "
    cPart += " AND B1_COD <= '" + MV_PAR02 + "' %"

    BeginSql alias cAlias

        SELECT B1_COD,B1_DESC,B1_LOCPAD,B1_POSIPI
        FROM %table:SB1% SB1
        WHERE B1_FILIAL = %xfilial:SB1%
        AND B1_MSBLQL <> '1'
        AND SB1.%notDel%

        %exp:cPart%

        ORDER BY B1_COD

    EndSql

    dbSelectArea(cAlias)
    (cAlias)->(dbGoTop())

    oReport:SetMeter((cAlias)->(LastRec()))

While !(cAlias)->( EOF() )

    If oReport:Cancel()
        Exit
    EndIf

    oReport:IncMeter()

    IncProc("Imprimindo " + alltrim((cAlias)->B1_DESC))

    //inicializo a primeira seção
    oSection1:Init()

    //imprimo a seção, relacionando os campos da section com os
    //valores da tabela

    oSection1:Cell("B1_COD"      ):SetValue((cAlias)->B1_COD      )
    oSection1:Cell("B1_DESC"     ):SetValue((cAlias)->B1_DESC     )
    oSection1:Cell("B1_LOCPAD"   ):SetValue((cAlias)->B1_LOCPAD   )
    oSection1:Cell("B1_POSIPI"   ):SetValue((cAlias)->B1_POSIPI   )
    oSection1:Printline()
```

```

(cAlias)->(dbSkip())

//imprimo uma linha
oReport:ThinLine()

Enddo

//finalizo seção
oSection1:Finish()

Return( NIL )

```

Relatório com método BeginQuery

```

User Function tReport_SQL()

Local oReport

//Criando o grupo de pergunta
CriaPerg()

//Carregando os dados da pergunta
Pergunte("XCLIVEND",.F.)

//Chando a Função para criar a estrutura do relatorio
oReport := ReportDef()

//Imprimindo o Relatorio
oReport:PrintDialog()

Return( Nil )
//*****
Static Function ReportDef()

Local oReport
Local oSection
Local oBreak

// Criando a o Objeto

oReport := TReport():New("ClinteXVendedor","Relatorio de Clientes X
Vendedor","XCLIVEND",{oReport| PrintReport(oReport)},"Relatorio de visitas de
vendedores nos clientes")

oSection := TRSection():New(oReport,"Dados dos Clientes",{ "SA1" } )

TRCell():New(oSection,"A1_VEND" , "SA1")
TRCell():New(oSection,"A1_COD" , " SA1","Cliente")
TRCell():New(oSection,"A1_LOJA" , "SA1")
TRCell():New(oSection,"A1_NOME" , "SA1")
TRCell():New(oSection,"A1_END" , "SA1")
TRCell():New(oSection,"A1_CEP" , "SA1")
TRCell():New(oSection,"A1_CONTATO", "SA1")
TRCell():New(oSection,"A1_TEL" , "SA1")
TRCell():New(oSection,"A1_EST" , "SA1")

```

```
// Quebra por Vendedor
oBreak := TRBreak():New(oSection,oSection:Cell("A1_EST"),,;
"Sub Total Estados")

//Fazendo a contagem por código
TRFunction():New(oSection:Cell("A1_COD" ), NIL, "COUNT", oBreak)

Return ( oReport )

//*****

Static Function PrintReport(oReport)
Local oSection := oReport:Section(1)
Local cPart := ""

oSection:BeginQuery()

    cPart := "% AND A1_COD >= '" + MV_PAR01 + "' "
    cPart += " AND A1_COD <= '" + MV_PAR02 + "' %"

BeginSql alias "QRYSA1"

    SELECT
        A1_COD,
        A1_LOJA,
        A1_NOME,
        A1_VEND,
        A1_ULTVIS,
        A1_TEMVIS,
        A1_TEL,
        A1_CONTATO,
        A1_EST
    FROM %table:SA1% SA1
    WHERE A1_FILIAL = %xfilial:SA1%
    AND A1_MSBLQL <> '1'
    AND SA1.%notDel%
    %exp:cPart%
    ORDER BY A1_EST Desc
EndSql

aRetSql := GetLastQuery()
oSection:EndQuery()
oSection:Print()

Return( Nil )

//=====
Static Function CriaPerg()

cPerg:= "XCLIVEND"

//Criação de pergunta no arquivo SX1
PutSx1( cPerg,"01","Cliente de";
,"","","mv_ch1","C",6,0,0,"M","","","SA1","","","MV_PAR01")
```

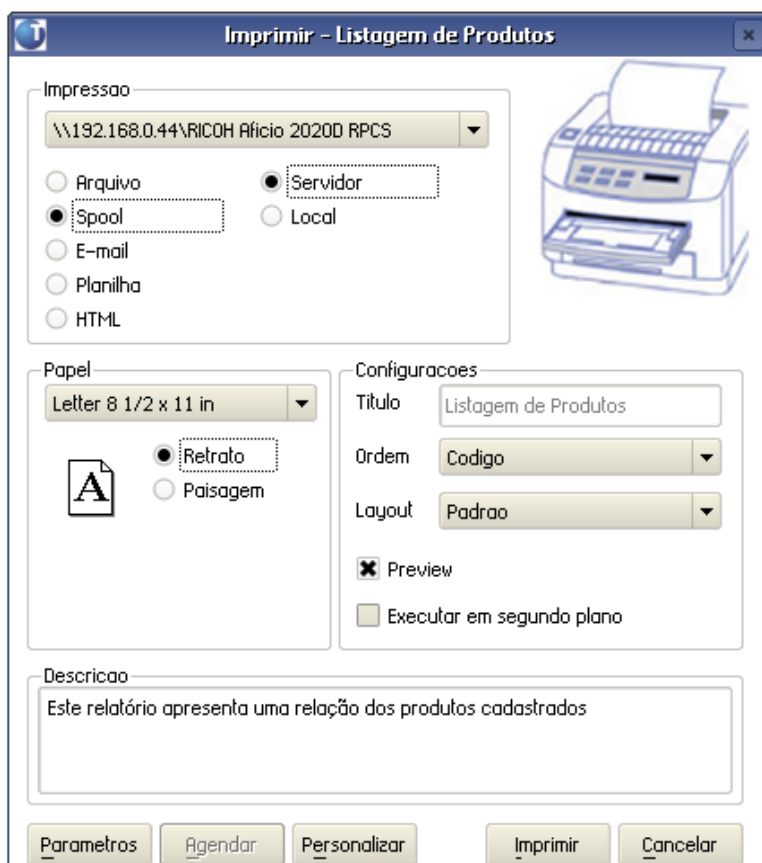
```
PutSx1( cPerg,"02","Cliente  
ate";, "", "", "mv_ch2", "C", 6, 0, 0, "M", "", "SA1", "", "", "MV_PAR02)
```

Verifique também o parâmetro MV_TReport. Para utilizar os relatórios personalizáveis, o parâmetro MV_TREPORT (tipo numérico) deve ser alterado no ambiente Configurador, conforme uma das opções que seguem:

- 1 = utiliza relatório no formato tradicional (antigo);
- 2 = utiliza relatório personalizável;
- 3 = pergunta qual relatório será utilizado: tradicional (antigo) ou personalizável

4.1.1. Impressão do relatório personalizável

Cada componente da tela de impressão do TReport, deve ser configurado no programa, para que o usuário tenha acesso às personalizações:



4.1.2. Parâmetros de impressão

A caixa de listagem apresentada deve ser utilizada conforme o meio de saída do relatório. Veja a seguir.

O relatório será gravado em disco com o nome apresentado. Caso seja escolhida a opção "Servidor" ele será gravado no diretório determinado na senha do usuário, através do configurador, sendo este sempre no servidor (padrão \SPOOL\). Na escolha da opção "Local" será aberta uma janela para que seja escolhido o local onde o relatório será gravado na máquina do usuário.

O relatório gerado a partir desta opção pode ser impresso ou enviado por e-mail após ser apresentado na tela.

Spool

Direciona o relatório para impressão via configuração do Windows® das impressoras instaladas.

E-mail

Envia o relatório por e-mail (Internet). Para isto, devem ser configurados os seguintes parâmetros no Ambiente Configurador:

- **MV_RELACNT**

Define a conta de e-mail para identificar a proveniência dos relatórios.

Exemplo: relprotheus@microsiga.com.br

- **MV_RELPSW**

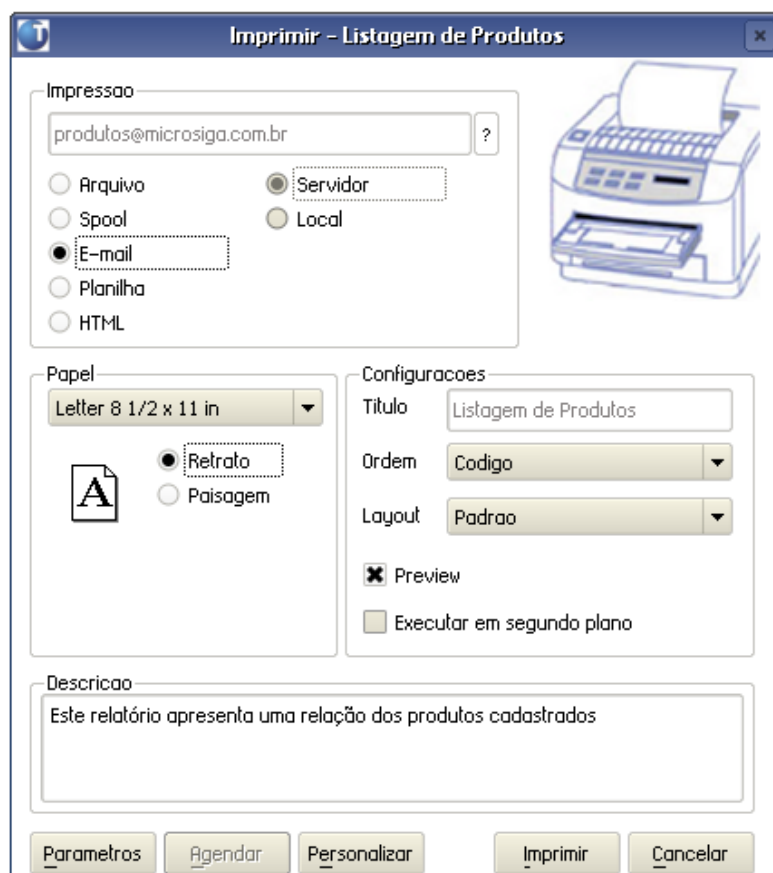
Define a senha da conta de e-mail para envio dos relatórios.

- **MV_RELSERV**

Define o servidor da conta de e-mail para o envio do relatório.

Exemplo: Mail.totvs.com.br

Quando selecionada esta opção, deve-se informar, no campo em destaque na figura abaixo, o e-mail para o qual o relatório deve ser remetido.



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

• Programação TelNet

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

Papel (Tamanho)

Selecione o tamanho do papel em que o relatório será impresso.

As especificações de tamanho do papel são as do padrão do mercado, conforme o formato escolhido, o Protheus irá ajustar a impressão.

Formato da impressão

Selecione o formato de impressão, clicando nos botões de opção “Retrato” ou “Paisagem”, fazendo assim que o relatório seja impresso na orientação vertical ou horizontal, respectivamente.

Título

Caso queira alterar a opção sugerida pelo sistema, digite o cabeçalho do relatório.

Ordem

Escolha a ordem em que as informações serão apresentadas no relatório, clicando em uma das chaves disponíveis.

Layout

Permite selecionar o modelo de relatório para impressão, à medida que novos leiautes forem gravados para um relatório, seus nomes serão listados nessa caixa.

Preview

Faz a exibição do relatório gerado na tela, possibilitando, na sequência, o seu envio para impressora ou a cancelamento da impressão.

Executar em segundo plano

Essa opção permite que o relatório seja gerado e enviado para a fila de impressão, enquanto o usuário pode executar outras tarefas no sistema.

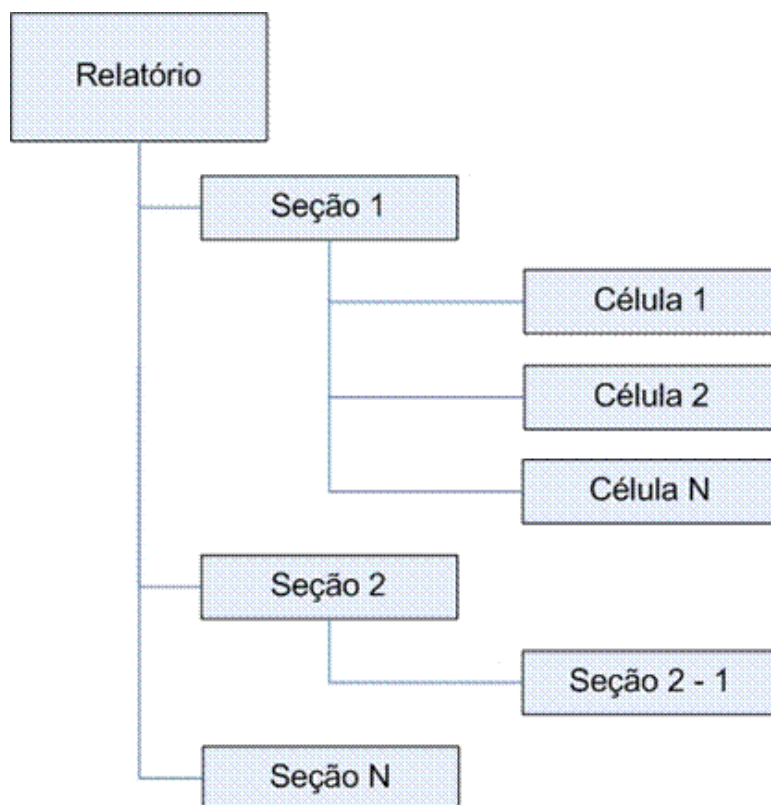
4.1.3. Personalização

É possível configurar-se as colunas do lay-out do relatório, bem como os acumuladores, cabeçalhos e linhas. Estão disponíveis para personalização também a fonte, tamanho, cores, e etc.

Editando o layout do relatório

O primeiro passo é entender a nova estrutura dos relatórios desenvolvidos com a ferramenta TReport.

O Relatório possui Seções e Células. É chamada de Seção, cada um dos grupos de informações, e de Célula, cada um dos campos que serão impressos. Nova estrutura do relatório TReport:



O relatório mais simples que se consegue emitir em TReport, é uma listagem.

4.1.4. Definindo a Função ReportDef()

A função ReportDef() é responsável pela construção do lay-out do relatório (oReport). É ela quem define as colunas, os campos e as informações que serão impressas.

Os comandos que fará essa construção são:

1. DEFINE REPORT
2. DEFINE SECTION
3. DEFINE CELL

DEFINE REPORT

A função DEFINE REPORT é responsável pela criação do objeto Report, ou seja, o relatório.

Internamente, o DEFINE REPORT irá executar o método TReport():New().

Estrutura do componente TReport:

⇒ O relatório (TReport) contém 1 ou mais seções (TRSection);

⇒ Uma seção (TRSection) pode conter 1 ou mais seções;

- ⇒ A seção (TRSection) contém células pré-definidas e células selecionadas pelo usuário;
 - ⇒ A seção (TRSection) também contém as quebras (TRBreak) para impressão de totalizadores (TRFunction);
- Os totalizadores são incluídos pela seção que automaticamente inclui no relatório (TRReport).

DEFINE SECTION

Ainda no ReportDef(), são definidas as seções (oSection) do relatório.

As [seções](#) do relatório representam os diferentes grupos de informações exibidos.

Há a seção principal e as específicas.

Internamente, o DEFINE SECTION irá executar o método TRSection():New().

A classe TRSection pode ser entendida como um layout do relatório, por conter células, quebras e totalizadores que darão um formato para sua impressão.

Com a classe TRSection é possível definir uma query, filtro ou índice com filtro (IndRegua) que será utilizada por ela para processamento do relatório, através do método Print e utilizando as células de posicionamento (TRPosition)

DEFINE CELL

Para cada seção, devem ser definidas as células. Célula é cada informação que deverá ser impressa. Pode ser um campo do cadastro, ou um resultado de uma operação. É uma Célula de impressão de uma seção (TRSection) de um relatório que utiliza a classe TRReport

Internamente, o DEFINE CELL irá executar o método TRCell():New ().

4.2. Funções Utilizadas para Desenvolvimento de Relatórios

4.2.1. Pergunte()

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

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

Parâmetros:

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

Retorno:

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

4.3. AjustaSX1()

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

Sintaxe: AJUSTASX1(cPerg, aPergs)

Parâmetros:

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

Estrutura – Item do array aPerg:

Posição	Campo	Tipo	Descrição
01	X1_PERGUNT	Caractere	Descrição da pergunta em português
02	X1_PERSPA	Caractere	Descrição da pergunta em espanhol
03	X1_PERENG	Caractere	Descrição da pergunta em inglês
04	X1_VARIAVL	Caractere	Nome da variável de controle auxiliar (mv_ch)
05	X1_TIPO	Caractere	Tipo do parâmetro
06	X1_TAMANHO	Numérico	Tamanho do conteúdo do parâmetro
07	X1_DECIMAL	Numérico	Número de decimais para conteúdos numéricos
08	X1_PRESEL	Numérico	Define qual opção do combo é a padrão para o parâmetro.
09	X1_GSC	Caractere	Define se a pergunta será do tipo G – Get ou C – Choice (combo)
10	X1_VALID	Caractere	Expressão de validação do parâmetro
11	X1_VAR01	Caractere	Nome da variável MV_PAR+“Ordem” do parâmetro
12	X1_DEF01	Caractere	Descrição da opção 1 do combo em português
13	X1_DEFSPA1	Caractere	Descrição da opção 1 do combo em espanhol
14	X1_DEFENG1	Caractere	Descrição da opção 1 do combo em inglês
15	X1_CNT01	Caractere	Conteúdo padrão ou último conteúdo definido como respostas para a pergunta.
16	X1_VAR02	Caractere	Não é informado
17	X1_DEF02	Caractere	Descrição da opção X do combo em português
18	X1_DEFSPA2	Caractere	Descrição da opção X do combo em espanhol
19	X1_DEFENG2	Caractere	Descrição da opção X do combo em inglês
20	X1_CNT02	Caractere	Não é informado
21	X1_VAR03	Caractere	Não é informado
22	X1_DEF03	Caractere	Descrição da opção X do combo em português
23	X1_DEFSPA3	Caractere	Descrição da opção X do combo em espanhol

24	X1_DEFENG3	Caractere	Descrição da opção X do combo em inglês
25	X1_CNT03	Caractere	Não é informado
26	X1_VAR04	Caractere	Não é informado
27	X1_DEF04	Caractere	Descrição da opção X do combo em português
28	X1_DEFSPA4	Caractere	Descrição da opção X do combo em espanhol
29	X1_DEFENG4	Caractere	Descrição da opção X do combo em inglês
30	X1_CNT04	Caractere	Não é informado
31	X1_VAR05	Caractere	Não é informado
32	X1_DEF05	Caractere	Descrição da opção X do combo em português
33	X1_DEFSPA5	Caractere	Descrição da opção X do combo em espanhol
34	X1_DEFENG5	Caractere	Descrição da opção X do combo em inglês
35	X1_CNT05	Caractere	Não é informado
36	X1_F3	Caractere	Código da consulta F3 vinculada ao parâmetro
37	X1_GRPSXG	Caractere	Código do grupo de campos SXG para atualização automática, quando o grupo for alterado.
38	X1_PYME	Caractere	Se a pergunta estará disponível no ambiente Pyme
39	X1_HELP	Caractere	Conteúdo do campo X1_HELP
40	X1_PICTURE	Caractere	Picture de formatação do conteúdo do campo.
41	aHelpPor	Array	Vetor simples contendo as linhas de help em português para o parâmetro. Trabalhar com linhas de até 40 caracteres.
42	aHelpEng	Array	Vetor simples contendo as linhas de help em inglês para o parâmetro. Trabalhar com linhas de até 40 caracteres.
43	aHelpSpa	Array	Vetor simples contendo as linhas de help em espanhol para o parâmetro. Trabalhar com linhas de até 40 caracteres.

4.3.1. PutSX1()

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

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

Parâmetros:

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

4.3.2. ParamBox()

Implementa uma tela de parâmetros, que não necessita da criação de um grupo de perguntas no SX1, e com funcionalidades que a Pergunte() não disponibiliza, tais como CheckBox e RadioButtons.

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

A PARAMBOX define os parâmetros seguindo o princípio das variáveis MV_PARxx. Caso ela seja utilizada em uma rotina em conjunto com parâmetros padrões (SX1 + Pergunte()) é necessário salvar os parâmetros padrões, chamar a Parambox(), salvar o retorno da Parambox() em variáveis Private específicas (MVPARBOXxx) e depois restaurar os parâmetros padrões, conforme o exemplo desta documentação.

Definir variáveis Private próprias para a ParamBox, as quais irão armazenar o conteúdo das MV_PARs que esta retorna.

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

Retorno: IOK à indica se a tela de parâmetros foi cancelada ou confirmada

aParametros : Array contendo os parâmetros

[1] Tipo do parâmetro (numérico) primeiro elemento não se altera, é sempre o tipo os demais se alteram conforme os tipos abaixo:

1. MsGet

- [2] : Descrição
- [3] : String contendo o inicializador do campo
- [4] : String contendo a Picture do campo
- [5] : String contendo a validação
- [6] : Consulta F3
- [7] : String contendo a validação When
- [8] : Tamanho do MsGet
- [9] : Flag .T./F. Parâmetro Obrigatório ?

2. Combo

- [2] : Descrição
- [3] : Numérico contendo a opção inicial do combo
- [4] : Array contendo as opções do Combo
- [5] : Tamanho do Combo
- [6] : Validação
- [7] : Flag .T./F. Parâmetro Obrigatório ?

3. Radio

- [2] : Descrição
- [3] : Numérico contendo a opção inicial do Radio
- [4] : Array contendo as opções do Radio
- [5] : Tamanho do Radio
- [6] : Validação
- [7] : Flag .T./F. Parâmetro Obrigatório ?
- [8] : String contendo a validação When

4. CheckBox (Com Say)

- [2] : Descrição
- [3] : Indicador Lógico contendo o inicial do Check
- [4] : Texto do CheckBox
- [5] : Tamanho do Radio
- [6] : Validação
- [7] : Flag .T./F. Parâmetro Obrigatório ?

5. CheckBox (linha inteira)

- [2] : Descrição
- [3] : Indicador Lógico contendo o inicial do Check
- [4] : Tamanho do Radio
- [5] : Validação
- [6] : Flag .T./F. Indica se campo é editável ou não

6. File

- [2] : Descrição
- [3] : String contendo o inicializador do campo
- [4] : String contendo a Picture do campo
- [5] : String contendo a validação
- [6] : String contendo a validação When
- [7] : Tamanho do MsGet
- [8] : Flag .T./F. Parâmetro Obrigatório ?
- [9] : Texto contendo os tipos de arquivo Ex.: "Arquivos .CSV |*.CSV"
- [10] : Diretório inicial do cGetFile
- [11] : PARAMETROS do cGETFILE

7. Montagem de expressão de filtro

- [2] : Descrição
- [3] : Alias da tabela
- [4] : Filtro inicial
- [5] : Opcional - Clausula When Botão Editar Filtro

8. MsGet Password

- [2] : Descrição
- [3] : String contendo o inicializador do campo
- [4] : String contendo a Picture do campo
- [5] : String contendo a validação
- [6] : Consulta F3
- [7] : String contendo a validação When
- [8] : Tamanho do MsGet
- [9] : Flag .T./F. Parâmetro Obrigatório ?

9. MsGet Say

- [2] : String Contendo o Texto a ser apresentado
- [3] : Tamanho da String
- [4] : Altura da String
- [5] : Negrito (lógico)

10. Range

- [2] : Descrição

- [3] : Range Inicial
- [4] : ConsultaF3
- [5] : Largo em pixels do Get
- [6] : Tipo
- [7] : Tamanho do campo (em chars)
- [8] : String contendo a validação When

11. MultiGet (Memo)

- [2] : Descrição
- [3] : Inicializador padrao
- [4] : String contendo a validação
- [5] : String contendo a validação When
- [6] : Flag .T./.F. Parâmetro Obrigatório ?

12. Filtro de usuário por Rotina

- [2] : Titulo do filtro
- [3] : Alias da tabela aonde será aplicado o filtro
- [4] : Filtro inicial
- [5] : Validação When

Parametros:

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

```
User Function ParamBox()
```

```
Local aRet      := {}
```

```
Local aParamBox := {}
```



```

Local aCombo := { "Janeiro",      ;
                  "Fevereiro",    ;
                  "Março",        ;
                  "Abril",        ;
                  "Maio",         ;
                  "Junho",        ;
                  "Julho",        ;
                  "Agosto",       ;
                  "Setembro",     ;
                  "Outubro",      ;
                  "Novembro",     ;
                  "Dezembro"}

Local i                := 0
Private cCadastro := "xParambox"

AADD(aParamBox,{1,"Produto",Space(15),"","","SB1","","0,.F.})
AADD(aParamBox,{2,"Tipo de cliente",1,aCombo,50,"",.F.})
AADD(aParamBox,{3,"Mostra deletados", ;
              IIF( Set(_SET_DELETED),1,2),{"Sim","Não"},50,"",.F.})

AADD(aParamBox,{4,"Marca todos ?",".F.,"Marque todos",50,"",.F.})
AADD(aParamBox,{5,"Marca todos ?",".F.",50,"",.F.})
AADD(aParamBox,{6,"Arquivo?",Space(50),"","","","50,.F.,"Arquivo .DBF
|*.DBF"})
AADD(aParamBox,{7,"Monte o
filtro","SX5","X5_FILIAL==xFilial('SX5')"})
AADD(aParamBox,{8,"Digite a senha",Space(15),"","","","80,.F.})

If ParamBox(aParamBox,"Teste ParamBox...",@aRet)
  For i := 1 To Len(aRet)
    MsgInfo(aRet[i],"Opção escolhida")
  Next
Endif

Return( NIL )

```

5. Utilizando Querys no Protheus

Podemos utilizar querys no Protheus quando acessamos bancos de dados via TopConnect. As querys, quando

bem construídas, melhoram enormemente a eficiência (velocidade) das consultas aos dados e reduzem a sobrecarga no servidor de aplicação, TopConnect e Banco de Dados. Normalmente uma query é utilizada em substituição a um Loop (While) na base de dados de programação convencional. Querys mais complexas utilizando joins poder ser construídas com a mesma função de vários loops.

O Protheus possui algumas funções:

TCQUERY()

Sintaxe	TCQUERY cSQL ALIAS cAlias NEW VIA "TOPCONN"
Descrição	<p>Executa uma Query no servidor e coloca seu retorno em uma WorkArea . Durante o processo de compilação</p> <p>cSQL: Expressão SQL a ser enviada ao servidor.</p> <p>ALIAS: cAlias especifica um nome para a WorkArea a ser aberta.</p> <p>NEW: Abre a tabela <cTable> na próxima WorkArea disponível. Se esta clausula não for especificada, <cTable> será na Work Área corrente.</p> <p>VIA: "TOPCONN" Este parâmetro indica ao ADVPL que esta Work Area será gerenciada pelo TOPconnect</p> <p>TCQUERY (cQuery) ALIAS (cAlias) NEW</p>

Durante o processo de compilação, a sintaxe TCQUERY() é substituída pelas expressão:

dbUseArea(.T., "TOPCONN", TcGenQry(, cQuery), "ALIAS" ,.T.,.F.)

Esta substituição é realizada conforme as definições do include TOPCONN.CH. Desta forma é recomendável a utilização direta da sintaxe DbUseArea() + TcGeQry().

```
Local cAlias := GetNextAlias()
Local cSql   := ""
```

```

cSql := " Select A2_COD,      "
cSql += "      A2_LOJA,      "
cSql += "      A2_NREDUZ,    "
cSql += "      A2_EST,       "
cSql += "      A2_MSBLQL     "
cSql += " FROM " + RetSQLName("SA2")
cSql += " Where A2_FILIAL = '" + xFilial('SA2') + "'"
cSql += " AND D_E_L_E_T_ = ' ' "

cSql := ChangeQuery(cSql)

TCQUERY ( cQuery ) ALIAS ( cAlias ) NEW

```

DbUseArea()

Sintaxe	DBUseArea(INovaArea,cDriver ,cTabela,cAlias,ICompartilhado , ISomenteLeitura)
Descrição	<p>LNovaArea: Caso verdadeiro, indica que a tabela deve ser aberta em uma nova workarea (Default=.F.) Caso o parâmetro seja .F. ou não especificado (NIL), a tabela será aberta na workarea atual, Caso já exista uma tabela aberta na WorkArea atual, a mesma é fechada.</p> <p>CDriver: Informa o Driver (RDD) a ser utilizada para a abertura da tabela. Caso não especificado (NIL), será usado o driver default de acesso a arquivos locais.</p> <p>cTabela: Nome da tabela a ser aberta. Caso o driver utilizado acesse tabelas no sistema de arquivos, deve ser informado um path no servidor de aplicação. Não é possível abrir tabelas de dados no SmartClient. Quando utilizada a RDD "TOPCONN", caso seja desejada a abertura de uma Query no SGDB, devemos passar como 3o parâmetro o retorno da função TcGenQry(), onde informamos a string contendo a Query como terceiro parâmetro da função TcGenQry()</p> <p>cAlias: Nome dado ao ALIAS desta tabela, para ser referenciado no programa Advpl.</p> <p>ICompartilhado: Caso verdadeiro, indica que a tabela deve ser aberta em modo compartilhado, isto é, outros processos também poderão abrir esta tabela.</p> <p>ISomenteLeitura: Caso verdadeiro, indica que este alias será usado apenas para leitura de dados. Caso contrário, estas operações serão perdidas</p>

TCGenQry (xPar1, xPar2, cQuery)

Permite a abertura de uma query diretamente no banco de dados utilizado na conexão atual, mediante uso da RDD TOPCONN. O retorno desta função deve ser passado como o 3o parâmetro da função DbUseArea() -- que corresponderia ao nome da tabela

```
Local cAliasFor := GetNextAlias()
Local cSql      := ""
cSql := " Select A2_COD,      "
cSql += "      A2_LOJA,      "
cSql += "      A2_NREDUZ,    "
cSql += "      A2_EST,       "
cSql += "      A2_MSBLQL     "
cSql += " FROM " + RetSQLName("SA2")
cSql += " Where A2_FILIAL = '" + xFilial('SA2') + "'"
cSql += " AND D_E_L_E_T_ = ' ' "

cSql := ChangeQuery(cSql)
dbUseArea( .T., "TOPCONN", TCGENQRY(, cSql), (cAliasFor), .F., .T.)
```

Funções de apoio:

- **CHANGEQUERY():** Função que efetua as adequações necessárias a query para que a mesma possa ser executada adequadamente no banco de dados em uso pela aplicação ERP através do TopConnect. Esta função é necessária pois a aplicação ERP Protheus pode ser utilizada com diversos bancos de dados, e cada banco possui particularidades em sua sintaxe, de forma que mesmo uma query escrita respeitando o padrão SQL ANSI podem necessitar de adequações.
Exemplo: CHANGEQUERY(cSql)
- **RetSQLName():** Retorna o nome padrão da tabela para seleção no banco de dados através da query.
Exemplo: RetSQLName("SA1")
- **xFilial():** Usada na construção da query para selecionar a filial corrente, da mesma forma que em ISAM.
Exemplo: xFilial("SA1")
- **TCSetField():** Compatibiliza os tipos de dados diferentes de caractere retornados pela query aos tipos do ADVPL.
Exemplo: TCSetField (< cAlias>, < cCampo>, < cTipo>, [nTamanho], [nDecimais])
TCSetField("cAlias", " D1_EMISSAO", "D")
TCSetField("cAlias", " D1_VUNIT " , "N", 12, 2)

GetNextAlias(): Retorna um alias para ser utilizado no record set definido em dbUseArea()

5.1. Embedded SQL

O objetivo do Embedded SQL é facilitar a escrita e leitura de queries. Para isso, foi definida uma sintaxe para que se possa escrever a query diretamente no código AdvPL, sem a necessidade de ficar concatenando pedaços de string para compor a string final.

O bloco de código onde será escrito o comando SELECT, deve sempre ser iniciado com BeginSQL Alias e finalizado com EndSQL.

Não é permitido incluir funções no meio do código embedded. Se precisar, o valor deve ser guardado em uma variável antes do início do BeginSQL.

```
Local cAliasProd := GetNextAlias()

BeginSql Alias cAlias
    Column D1_QUANT as Numeric(12,2)
    Column D1_EMISSAO as date
    %NOPARSER%
    SELECT D1_DOC,
           D1_FORNECE,
           D1_LOJA,
           D1_Serie,
           D1_QUANT,
           D1_ITEM,
           D1_EMISSAO
    FROM   %TABLE:SD1%
    WHERE  D1_FILIAL = %xFilial:SD1%
           AND D1_EMISSAO >= %EXP:pDatade%
           AND D1_EMISSAO <= %EXP:pDataAte%
           AND %NOTDEL%

EndSql
```

Características operacionais – Sintaxe

%Table:SB1% : Retorna o nome padrão da tabela para seleção no banco de dados através da query.

%NotDel% : é substituída por D_E_L_E_T_=' '.

%xFilial:SB1% é substituída pela função xFilial("SB1")

%exp:%: Para informar uma variável no "Select" necessário informar %exp:NOME_DA_VARIAVEL%

Column: Faz a transformação do campo igual **TCSetField()**

%NOPARSER%: Indica que a query não deve passar pela função ChangeQuery() antes de ser enviada ao banco de dados. Caso não especificado, o padrão é a string da query ser passada automaticamente pela função ChangeQuery().

6. Manipulação de arquivos

6.1. Geração e leitura de arquivos em formato texto

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

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

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

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

2ª) Família: nesta família serão utilizadas as funções: FT_FUse(), FT_FGoTop(), FT_FLastRec(), FT_FEOF(), FT_FReadLn(), FT_FSkip(), FT_FGoto(), FT_FRecno().

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

6.2. 1ª Família de funções de gravação e leitura de arquivos texto

6.2.1. FCREATE()

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

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

Atributos definidos no include FileIO.ch

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

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

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

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

Parâmetros:

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

Retorno:

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

6.2.2. FWRITE()

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

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

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

Parâmetros:

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

Retorno:

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

6.2.3. FCLOSE()

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

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

Sintaxe: FCLOSE (< nHandle >)

Parâmetros:

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

Retorno:

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

6.2.4. FSEEK()

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

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

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

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

Parâmetros:

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

Retorno:

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

6.2.5. FOPEN()

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

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

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

Parâmetros:

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

Retorno:

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

6.2.6. FREAD()

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

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

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

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

Parâmetros:

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

Retorno:

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

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

```
#include "protheus.ch"

User Function GeraTXT()

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

//CriaSx1(cPerg)
//Pergunte(cPerg,.F.)
dbSelectArea(cAlias)
dbSetOrder(1)
//+-----+
//| Montagem da tela de processamento.|
//+-----+
DEFINE MSDIALOG oGeraTxt TITLE OemToAnsi("Geração de Arquivo Texto") ;
FROM 000,000 TO 200,400 PIXEL

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

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

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

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

ACTIVATE DIALOG oGeraTxt CENTERED

Return Nil

/*/
+-----+
| Função      | OKGERATXT | Autor | SERGIO FUZINAKA | Data |
+-----+
| Descrição   | Função chamada pelo botão OK na tela inicial de
```

```
|processamento.
| Executa a geração do arquivo texto.
+-----+
/*/

Static Function OkGeraTxt

//+-----+//|
Cria o arquivo texto
//+-----+

Private cArqTxt := "\SYSTEM\EXPSA1.TXT"
Private nHdl     := fCreate(cArqTxt)

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

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

Return Nil

//-----+

Static Function RunCont

Local cLin

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

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

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

//+-----+
//| Gravação no arquivo texto. Testa por erros durante a gravação da |
//| linha montada. |
//+-----+

If fWrite(nHdl,cLin,Len(cLin)) != Len(cLin)
    If !MsgAlert("Ocorreu um erro na gravação do arquivo."+
    "Continua?", "Atenção!")

        Exit
    Endif
Endif
```

```

Endif

        (cAlias)->(dbSkip())
EndDo

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

Return Nil

```

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

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

```

#include "protheus.ch"
User Function LeTXT()

//+-----+
//| Declaração de Variáveis |
//+-----+

Local cPerg      := "IMPSA1"
Local oLeTxt

Private cAlias := "SA1"

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

dbSelectArea(cAlias)
dbSetOrder(1)

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

DEFINE MSDIALOG oLeTxt TITLE OemToAnsi("Leitura de Arquivo Texto");
FROM 000,000 TO 200,400 PIXEL
@ 005,005 TO 095,195 OF oLeTxt PIXEL
@ 10,020 Say " Este programa ira ler o conteúdo de um arquivo texto, conforme";
OF oLeTxt PIXEL
@ 18,020 Say " os parâmetros definidos pelo usuário, com os registros do
arquivo";
OF oLeTxt PIXEL
@ 26,020 Say " SA1" OF oLeTxt PIXEL

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

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

DEFINE SBUTTON FROM 070, 110 TYPE 5 ;

```

```

ACTION (Pergunte(cPerg,.T.)) ENABLE OF oLeTxt
ACTIVATE DIALOG oLeTxt CENTERED

Return Nil

Static Function OkLeTxt()

Private cArqTxt := "\SYSTEM\EXPSA1.TXT"
Private nHdl    := fOpen(cArqTxt,68)

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

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

Static Function RunCont

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

//1234567890123456789012345678901234567890123456789012345678901234567890
//000000000010000000002000000000300000000040000000005000000000600000000070
//FFCCCCCLNNNNNNNNNNNNNNNNNNNNNNNNNNNNVVVVVVVVVVVVVVVVDDDDDDDD
//A1_FILIAL      - 01, 02 - TAM: 02
//A1_COD         - 03, 08 - TAM: 06
//A1_LOJA        - 09, 10 - TAM: 02
//A1_NREDUZ      - 11, 30 - TAM: 20
//A1_MCOMPRA     - 31, 46 - TAM: 14,2
//A1_ULTCOM      - 47, 54 - TAM: 08

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

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

//Incrementa a régua
IncProc()

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

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

```

While .T.
    IF dbSeek(cFilSA1+cCodSA1+cLojaSA1)
        cCodSA1 := SOMA1(cCodSA1)
        Loop
    Else
        Exit
    Endif
Enddo
dbSelectArea(cAlias)
RecLock(cAlias,.T.)
(cAlias)->A1_FILIAL      := cFilSA1  //- 01, 02 - TAM: 02
(cAlias)->A1_COD         := cCodSA1  //- 03, 08 - TAM: 06
(cAlias)->A1_LOJA        := cLojaSA1 //- 09, 10 - TAM: 02
(cAlias)->A1_NREDUZ      := Substr(cBuffer,11,20)
//- 11, 30 - TAM: 20
(cAlias)->A1_MCOMPRA     := Val(Substr(cBuffer,31,16))/100
//- 31, 46 - TAM: 14,2
(cAlias)->A1_ULTCOM      := STOD(Substr(cBuffer,47,08))
//- 47, 54 - TAM: 08
MSUnlock()

EndDo

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

Return Nil

```

6.3. 2ª Família de funções de gravação e leitura de arquivos texto

6.3.1. FT_FUSE()

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

Sintaxe: FT_FUSE ([cTXTFile])

Parâmetros:

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

Retorno:

Numérico

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

6.3.2. FT_FGOTOP()

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

Sintaxe: FT_FGOTO (< nPos >)

Parâmetros:

nPos

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

6.3.3. FT_FLASTREC()

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

Sintaxe: FT_FLASTREC()

Parâmetros:

Nenhum

.

Retorno:

Numérico

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

6.3.4. FT_FEOF()

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

Sintaxe: FT_FEOF()

Parâmetros:

Nenhum

.

Retorno:

Lógico

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

6.3.5. FT_FREADLN()

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

Sintaxe: FT_FREADLN()**Parâmetros:****Nenhum**

.

Retorno:**Caracter**

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

6.3.6. FT_FSKIP()

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

Sintaxe: FT_FSKIP ([nLinhas])**Parâmetros:****nLinhas**

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

Retorno:**Nenhum**

.

6.3.7. FT_FGOTO()

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

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

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

Retorno:

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

6.3.8. FT_FREQNO()

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

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

Sintaxe: FT_FREQNO ()

Parâmetros:

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

Retorno:

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

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

```
#Include "Protheus.ch"

User Function LeArqTxt()

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

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

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

FormBatch( cCadastro, aSay, aButton )

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

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

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

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

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

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

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

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

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

    MsgInfo(cMsg)

    FT_FSKIP() //próximo registro no arquivo txt
EndDo

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

Exercício

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

7. Captura de múltiplas informações (Multi-Lines)

A linguagem ADVPL permite a utilização de basicamente dois tipos de objetos do tipo grid, ou como também são conhecidos: multi-line:

- **Grids digitáveis:** permitem a visualização e captura de informações, comumente utilizados em interfaces de cadastro e manutenção, tais como:

 **MSNEWGETDADOS()**

- **Grids não digitáveis:** permitem somente a visualização de informações, comumente utilizados como browses do ERP Protheus, tais como:

 **TCBROWSER()**

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

7.1.1. MsNewGetDados()

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

Sintaxe: MsNewGetDados():New(nSuperior, nEsquerda, nInferior, nDireita, nOpc, cLinOk, cTudoOk, cIniCpos, aAlterGDa, nFreeze, nMax, cFieldOk, cSuperDel, cDelOk, oDLG, aHeader, aCols)

Retorno: oMsGetDados → objeto do tipo MsNewGetDados()

Parâmetros:

nSuperior	Distância entre a MsNewGetDados e o extremidade superior do objeto que a contém
nEsquerda	Distância entre a MsNewGetDados e o extremidade esquerda do objeto que a contém
nInferior	Distância entre a MsNewGetDados e o extremidade inferior do objeto que a contém
nDireita	Distância entre a MsNewGetDados e o extremidade direita do objeto que a contém
nOpc	Operação em execução: 2- Visualizar, 3- Incluir, 4- Alterar, 5- Excluir
cLinOk	Função executada para validar o contexto da linha atual do aCols
cTudoOk	Função executada para validar o contexto geral da MsNewGetDados (todo aCols)
clniCpos	Nome dos campos do tipo caracter que utilizarão incremento automático.
aAlterGDa	Campos alteráveis da GetDados
nFreeze	Campos estáticos na GetDados, partindo sempre da posição inicial da getdados aonde: <ul style="list-style-type: none"> 1- Primeiro campo congelado 2- Primeiro e segundo campos congelados...
nMax	Número máximo de linhas permitidas. Valor padrão 99
cFieldOk	Função executada na validação do campo
cSuperDel	Função executada quando pressionada as teclas <Ctrl>+<Delete>
cDelOk	Função executada para validar a exclusão de uma linha do aCols
oDLG	Objeto no qual a MsNewGetDados será criada
aHeader	Array a ser tratado internamente na MsNewGetDados como aHeader
aCols	Array a ser tratado internamente na MsNewGetDados como aCols

Variáveis private:

aRotina	<p>Vetor com as rotinas que serão executadas na MBrowse e que definirá o tipo de operação que esta sendo executada (inclusão, alteração, exclusão, visualização, pesquisa, ...) no formato:</p> <p style="text-align: center;">{cTitulo, cRotina, nOpção, nAcesso}, aonde:</p> <p>nOpção segue o padrão do ERP Protheus para:</p> <ul style="list-style-type: none"> 1- Pesquisar 2- Visualizar 3- Incluir 4- Alterar 5- Excluir
aHeader	<p>Vetor com informações das colunas no formato:</p> <p>{cTitulo, cCampo, cPicture, nTamanho, nDecimais,;</p>

	cValidação, cReservado, cTipo, xReservado1, xReservado2}
	A tabela temporária utilizada pela MsGetDB deverá ser criada com base no aHeader mais um último campo tipo lógico que determina se a linha foi excluída.
IRefresh	Variável tipo lógica para uso reservado.

Variáveis públicas:

N	Indica qual a linha posicionada do aCols.
---	---

Funções de validação:

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

Métodos adicionais:

ForceRefresh()	Atualiza a MsNewGetDados com a tabela e posiciona na primeira linha.
Hide()	Oculto a MsNewGetDados
Show()	Mostra a MsNewGetDados.

Exemplo: Utilização dos objetos MsNewGetDados() e MsMGet()

```
#include "protheus.ch"

User Function MrbwGetD()

Private cCadastro      := "Pedidos de Venda"
Private aRotina := {{"Pesquisar"      , "axPesqui"      , 0, 1},;
                   {"Visualizar"     , "U_ModGtd"     , 0, 2},;
                   {"Incluir"        , "U_ModGtd"     , 0, 3}}

DbSelectArea("SC5")
DbSetOrder(1)

MBrowse(6,1,22,75,"SC5")

Return
//-----
User Function ModGtd(cAlias,nReg,nOpc)

Local nX      := 0
Local nUsado  := 0
Local aButtons := {}
Local aCpoEnch := {}
```

```

Local cAliasE      := cAlias
Local aAlterEnch    := {}
Local aPos          := {000,000,080,400}
Local nModelo       := 3
Local lF3           := .F.
Local lMemoria       := .T.
Local lColumn       := .F.
Local caTela        := ""
Local lNoFolder     := .F.
Local lProperty     := .F.
Local aCpoGDa       := {}
Local cAliasGD      := "SC6"
Local nSuperior     := 081
Local nEsquerda     := 000
Local nInferior     := 250
Local nDireita      := 400
Local cLinOk        := "AllwaysTrue"
Local cTudoOk       := "AllwaysTrue"
Local cIniCpos      := "C6_ITEM"
Local nFreeze       := 000
Local nMax          := 999
Local cFieldOk      := "AllwaysTrue"
Local cSuperDel     := ""
Local cDelOk        := "AllwaysFalse"
Local aHeader       := {}
Local aCols         := {}
Local aAlterGDa := {}

Private oDlg
Private oGetD
Private oEnch
Private aTELA[0][0]
Private aGETS[0]

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

While !Eof() .And. SX3->X3_ARQUIVO == cAliasE
    If !(SX3->X3_CAMPO $ "C5_FILIAL") .And. cNivel >= SX3->X3_NIVEL .And.;
        X3Uso(SX3->X3_USADO)
        AADD(aCpoEnch, SX3->X3_CAMPO)
    EndIf
    DbSkip()
EndDo

aAlterEnch := aClone(aCpoEnch)

DbSelectArea("SX3")
DbSetOrder(1)
MsSeek(cAliasGD)

While !Eof() .And. SX3->X3_ARQUIVO == cAliasGD
    If ! (AllTrim(SX3->X3_CAMPO) $ "C6_FILIAL") .And.;
        cNivel >= SX3->X3_NIVEL .And. X3Uso(SX3->X3_USADO)
        AADD(aCpoGDa, SX3->X3_CAMPO)
    EndIf
    DbSkip()

```

```

EndDo

aAlterGDa := aClone(aCpoGDa)

nUsado:=0
dbSelectArea("SX3")
dbSeek("SC6")
aHeader:={}
While !Eof().And.(x3_arquivo=="SC6")
    If X3USO(x3_usado).And.cNivel>=x3_nivel
        nUsado:=nUsado+1
    AADD(aHeader,{ TRIM(x3_titulo), x3_campo, x3_picture,x3_tamanho,;
x3_decimal,"AlwaysTrue()",x3_usado, x3_tipo, x3_arquivo, x3_context })
    Endif
    dbSkip()
End

If nOpc==3 // Incluir
    aCols:={Array(nUsado+1)}
    aCols[1,nUsado+1]:=F.
    For nX:=1 to nUsado
        IF aHeader[nX,2] == "C6_ITEM"
            aCols[1,nX]:= "0001"
        ELSE
            aCols[1,nX]:=CriaVar(aHeader[nX,2])
        ENDIF
    Next
Else
    aCols:={}
    dbSelectArea("SC6")
    dbSetOrder(1)
    dbSeek(xFilial()+M->C5_NUM)
    While !eof().and.C6_NUM==M->C5_NUM
        AADD(aCols,Array(nUsado+1))
        For nX:=1 to nUsado
            aCols[Len(aCols),nX]:=FieldGet(FieldPos(aHeader[nX,2]))
        Next
        aCols[Len(aCols),nUsado+1]:=F.
        dbSkip()
    End
Endif

oDlg      := MSDIALOG():New(000,000,400,600, cCadastro,,,,,,,,.T.)
RegToMemory("SC5", If(nOpc==3,.T.,.F.))

oEnch := MsMGet():New(cAliasE,nReg,nOpc,/*aCRA*/,/*cLetra*/,/*cTexto*/,;
    aCpoEnch,aPos,aAlterEnch, nModelo, /*nColMens*/, /*cMensagem*/,;
    /*cTudoOk*/, oDlg,lF3, lMemoria,lColumn,caTela,lNoFolder,;
    lProperty)

oGetD:= MsNewGetDados():New(nSuperior, nEsquerda, nInferior, nDireita,;
    nOpc,cLinOk,cTudoOk, cIniCpos, aAlterGDa, nFreeze, nMax,cFieldOk,;
    cSuperDel,cDelOk, oDLG, aHeader, aCols)

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


7.1.2. Definindo cores personalizadas para o objeto MsNewGetDados()

Conforme visto no tópico sobre definição das propriedades de cores para os componentes visuais, cada objeto possui características que devem ser respeitadas para correta utilização deste recurso.

Atributos adicionais:

IUseDefaultColors

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

Métodos adicionais:

SetBlkBackColor

Método que define a cor que será utilizada para cada linha do grid. Não é necessário utilizar o método Refresh() após a definição da cor por este método.

Exemplo: Definindo cores personalizadas para o objeto MsNewGetDados()

```
#include "protheus.ch"

User Function MrbwGtCl()

Private cCadastro      := "Pedidos de Venda"
Private aRotina := {{"Pesquisar"      , "axPesqui"      , 0, 1},;
                   {"Visualizar"     , "U_ModGtd"     , 0, 2},;
                   {"Incluir"        , "U_ModGtd"     , 0, 3}}

DbSelectArea("SC5")
DbSetOrder(1)

MBrowse(6,1,22,75,"SC5")

Return

User Function ModGtd(cAlias,nReg,nOpc)

Local nX      := 0
Local nUsado  := 0

Local aButtons := {}
Local aCpoEnch := {}
Local cAliasE   := cAlias
Local aAlterEnch := {}
Local aPos      := {000,000,080,400}
Local nModelo   := 3
Local lF3       := .F.
Local lMemoria  := .T.
Local lColumn   := .F.
Local caTela    := ""
Local lNoFolder := .F.
Local lProperty := .F.
Local aCpoGDa   := {}
Local cAliasGD  := "SC6"
```

```

Local nSuperior      := 081
Local nEsquerda      := 000
Local nInferior      := 250
Local nDireita       := 400
Local cLinOk         := "AllwaysTrue"
Local cTudoOk        := "AllwaysTrue"
Local cIniCpos       := "C6_ITEM"
Local nFreeze        := 000
Local nMax           := 999
Local cFieldOk       := "AllwaysTrue"
Local cSuperDel      := ""
Local cDelOk         := "AllwaysFalse"
Local aHeader        := {}
Local aCols          := {}
Local aAlterGDa      := {}

Private oDlg
Private oGetD
Private oEnch
Private aTELA[0][0]
Private aGETS[0]

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

While !Eof() .And. SX3->X3_ARQUIVO == cAliasE
    If !(SX3->X3_CAMPO $ "C5_FILIAL") .And. cNivel >= SX3->X3_NIVEL .And.;
    X3Uso(SX3->X3_USADO)
        AADD(aCpoEnch, SX3->X3_CAMPO)
    EndIf
    DbSkip()
End

aAlterEnch := aClone(aCpoEnch)

DbSelectArea("SX3")
DbSetOrder(1)
MsSeek(cAliasGD)

While !Eof() .And. SX3->X3_ARQUIVO == cAliasGD
    If !(AllTrim(SX3->X3_CAMPO) $ "C6_FILIAL") .And. cNivel >= SX3->X3_NIVEL
    .And. X3Uso(SX3->X3_USADO)
        AADD(aCpoGDa, SX3->X3_CAMPO)
    EndIf
    DbSkip()
End

aAlterGDa := aClone(aCpoGDa)

nUsado:=0
dbSelectArea("SX3")
dbSeek("SC6")
aHeader:={}
While !Eof().And.(x3_arquivo=="SC6")
    If X3USO(x3_usado).And.cNivel>=x3_nivel
        nUsado:=nUsado+1
    AADD(aHeader,{ TRIM(x3_titulo), x3_campo, x3_picture,;

```

```

        x3_tamanho, x3_decimal, "AlwaysTrue()", ;
        x3_usado, x3_tipo, x3_arquivo, x3_context } )
    Endif
    dbSkip()
End

If nOpc==3 // Incluir
    aCols:={Array(nUsado+1)}
    aCols[1,nUsado+1]:=F.
    For nX:=1 to nUsado

        IF aHeader[nX,2] == "C6_ITEM"
            aCols[1,nX] := "0001"
        ELSE
            aCols[1,nX]:=CriaVar(aHeader[nX,2])
        ENDIF

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

oDlg := MSDIALOG():New(000,000,400,600, cCadastro,,,,,,,,.T.)

RegToMemory("SC5", If(nOpc==3,.T.,.F.))

oEnch := MsMGet():New(cAliasE,nReg,nOpc,/*aCRA*/,/*cLetra*/, /*cTexto*/,;
    aCpoEnch,aPos, aAlterEnch, nModelo, /*nColMens*/, /*cMensagem*/,;
    cTudoOk,oDlg,lF3, lMemoria,lColumn,caTela,lNoFolder,lProperty)

oGetD:= MsNewGetDados():New(nSuperior,nEsquerda,nInferior,nDireita, nOpc,;
    cLinOk,cTudoOk,cIniCpos,aAlterGDa,nFreeze,nMax,cFieldOk, cSuperDel,;
    cDelOk, oDLG, aHeader, aCols)

// Tratamento para definição de cores específicas,
// logo após a declaração da MsNewGetDados

oGetD:oBrowse:lUseDefaultColors := .F.
oGetD:oBrowse:SetBlkBackColor({|| GETDCLR(oGetD:aCols,oGetD:nAt,aHeader)})

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

Return

```

```
// Função para tratamento das regras de cores para a grid da MsNewGetDados

Static Function GETDCLR(aLinha,nLinha,aHeader)

Local nCor2      := 16776960 // Ciano - RGB(0,255,255)
Local nCor3      := 16777215 // Branco - RGB(255,255,255)
Local nPosProd   := aScan(aHeader,{|x| Alltrim(x[2]) == "C6_PRODUTO"})
Local nUsado     := Len(aHeader)+1
Local nRet       := nCor3

If !Empty(aLinha[nLinha][nPosProd]) .AND. aLinha[nLinha][nUsado]
    nRet := nCor2
ElseIf !Empty(aLinha[nLinha][nPosProd]) .AND. !aLinha[nLinha][nUsado]
    nRet := nCor3
Endif
Return nRet
```

7.2. TCBrowse

A classe de objetos visuais TCBrowse() permite a criação de um grid com uma ou mais colunas, baseado na sua estrutura

- Metodo Contrutor: **TCBrowse:New**

Propriedades:

- **TCBrowse:aArray**: Indica o array que contém as informações apresentadas no browse.
- **TCBrowse:aColBmps**: Indica um array com campos lógicos para determinar se a coluna é ou não uma imagem.
- **TCBrowse:aColSizes**: Indica a largura das colunas.
- **TCBrowse:aColumns**: Indica um array com objetos da classe TCColumn (caso tenha utilizado este componente para inserir colunas).
- **TCBrowse:aHeaders**: Indica o título dos campos no cabeçalho.
- **TCBrowse:bBmpName**: Indica o bloco de código que será executado, internamente pelo browse, quando trocar a imagem de uma célula.
- **TCBrowse:bDelOk**: Indica o bloco de código que será executado quando excluir uma linha do browse.
- **TCBrowse:bDrawSelect**: Indica o bloco de código que será executado ao utilizar o teclado para mudar de linha.
- **TCBrowse:bGoBottom**: Indica o bloco de código que será executado quando o método GoBottom() for chamado.
- **TCBrowse:bGoTop**: Indica o bloco de código que será executado quando o método GoTop() é chamado.
- **TCBrowse:bHeaderClick**: Indica o bloco de código que será executado quando clicar, com o botão esquerdo do mouse, no cabeçalho do browse.
- **TCBrowse:blnRange**: Indica o bloco de código que será executado para avaliar filtro no registro corrente. Indica o bloco de código que será executado quando clicar duas vezes, com o botão esquerdo do mouse, sobre o objeto.
- **TCBrowse:blDbClick**:
- **TCBrowse:bLine**: Indica o bloco de código que será executado para montar a linha do browse.
- **TCBrowse:bLogicLen**: Indica o bloco de código que será executado para contar as linhas do browse.
- **TCBrowse:bSeekChange**: Indica o bloco de código que será executado quando mudar de linha.
- **TCBrowse:bSkip**: Indica o bloco de código que será executado quando mudar de linha.
- **TCBrowse:bSuperDel**: Indica o bloco de código que será executado quando excluir uma linha do browse.
- **TCBrowse:bValid**: Indica o bloco de código de validação que será executado quando o conteúdo do objeto for modificado. Retorna verdadeiro (.T.), se o conteúdo é válido; caso contrário, falso (.F.).

- **TCBrowse:cAlias:** Indica se o objeto é utilizado com array ou tabela.
- **TCBrowse:cField:** Indica as propriedades relacionadas com o filtro.
- **TCBrowse:cOrderType:** Indica o tipo de ordenação corrente. Exemplo: "D" é igual a ordenação por campo data.
- **TCBrowse:cSeek:** Indica a chave de pesquisa incremental.
- **TCBrowse:IAdjustColSize:** Indica se, verdadeiro (.T.), permite ajustar a largura da célula.
- **TCBrowse:IAutoEdit:** Compatibilidade. Propriedade implementada somente na classe MsBrGetDBase.
- **TCBrowse:IDisablePaint:** Indica se, verdadeiro (.T.), desabilita a atualização do browse.
- **TCBrowse:IHitBottom:** Indica o tipo de ponteiro do mouse que está na última linha do browse.
- **TCBrowse:IHitTop:** TCBrowse:IHitTop
- **TCBrowse:IHScroll:** TCBrowse:IHScroll
- **TCBrowse:IJustific:** Indica se, verdadeiro (.T.), centraliza o texto ou, falso (.F.), alinha o texto à esquerda.
- **TCBrowse:IUseDefaultColors:** Indica se, verdadeiro (.T.), utiliza as cores padrão do browse.
- **TCBrowse:IVScroll:** Indica se, verdadeiro (.T.), habilita a barra de rolagem vertical.
- **TCBrowse:nAt:** Retorna a linha selecionada.
- **TCBrowse:nColOrder** Indexa os dados exibidos no browse quando utilizado um alias (tabela no banco de dados).
- **TCBrowse:nColPos:** Indica o posicionamento da coluna no browse.
- **TCBrowse:nFreeze:** Indica a coluna que será congelada à esquerda. Só é permitido o congelamento de uma coluna, qualquer valor maior que 1 será convertido para 1.
- **TCBrowse:nLen:** Indica o número total de linhas.
- **TCBrowse:nLinhas:** Indica o número de linhas por célula na vertical.
- **TCBrowse:nRowPos:** Indica o posicionamento da linha no browse.
- **TCBrowse:nScrollType:** Indica o tipo da barra de rolagem que será utilizada.

Métodos:

- **TCBrowse:AddColumn:** Inclui coluna no browse.
- **TCBrowse:ColPos:** Retorna o número da coluna posicionada.
- **TCBrowse:DrawSelect::** Força a atualização do browse.
- **TCBrowse:GetBrowse:** Retorna o objeto da classe TCBrowse.
- **TCBrowse:GetBrwOrder:** Retorna um array com os títulos, definidos pelo usuário, das colunas.
- **TCBrowse:GetCellRect:** Retorna o retângulo da célula, do browse, no formato da classe TRect.
- **TCBrowse:GetColSizes:** Retorna um array com as larguras das colunas.
- **TCBrowse:GoBottom:** Move o ponteiro do mouse para a primeira linha do browse.
- **TCBrowse:GoColumn:** Posiciona o cursor na coluna desejada.
- **TCBrowse:GoDown:** Move o ponteiro do mouse uma célula abaixo.
- **TCBrowse:GoLeft:** Move o ponteiro do mouse para a célula adjacente à esquerda.
- **TCBrowse:GoPosition:** Posiciona o ponteiro do mouse na linha desejada.
- **TCBrowse:GoRight:** Move o ponteiro do mouse para a célula adjacente à direita.
- **TCBrowse:GoTop:** Move o ponteiro do mouse para a primeira linha do browse.
- **TCBrowse:GoUp:** Move o ponteiro do mouse uma célula acima.
- **TCBrowse:nAtCol:** Retorna a coluna em uma determinada posição do browse.
- **TCBrowse:nRowCount:** Retorna o número de linhas que estão visíveis no browse.
- **TCBrowse:PageDown:** Move o ponteiro do mouse para baixo, conforme o número de linha configurado.
- **TCBrowse:PageUp:** Move o ponteiro do mouse para cima, conforme o número de linha configurado.
- **TCBrowse:ResetLen:** Reinicia o contador de linha do browse.
- **TCBrowse:SetArray:** Define um array para o browse.
- **TCBrowse:SetBlkBackColor:** Define a cor de fundo das colunas.
- **TCBrowse:SetBlkColor:** Define a cor de fundo das colunas.

- **TCBrowse:SetFilter:** Define a cor da fonte das colunas.
- **TCBrowse:setHeaderImage:** Define o filtro para os registros do browse.
- **TCBrowse:Skip:** Posiciona o ponteiro do mouse "n" linhas para frente.

Exemplo:

```
#include "TOTVS.CH"

USER FUNCTION TCBrowse()
Local oOK := LoadBitmap(GetResources(),'br_verde')
Local oNO := LoadBitmap(GetResources(),'br_vermelho')
Local aList := {}

    DEFINE DIALOG oDlg TITLE "Exemplo TCBrowse" FROM 180,180 TO 550,700 PIXEL

        // Vetor com elementos do Browse
        aBrowse := { {T.,'CLIENTE 001','RUA CLIENTE 001',111.11},,
                    {F.,'CLIENTE 002','RUA CLIENTE 002',222.22},,
                    {T.,'CLIENTE 003','RUA CLIENTE 003',333.33} }

        // Cria Browse
        oBrowse := TCBrowse():New( 01 , 01, 260, 156,,
                                   {'','Codigo','Nome','Valor'},{20,50,50,50},,
                                   oDlg,,,,{||},,,,,F,,,T,,,F,,,, )

        // Seta vetor para a browse
        oBrowse:SetArray(aBrowse)

        // Monta a linha a ser exibida no Browse
        oBrowse:bLine := {||{ If(aBrowse[oBrowse:nAt,01],oOK,oNO),,
                               aBrowse[oBrowse:nAt,02],,
                               aBrowse[oBrowse:nAt,03],,
                               Transform(aBrowse[oBrowse:nAt,04],'@E 99,999,999,999.99') } }

        // Evento de clique no cabeçalho da browse
        oBrowse:bHeaderClick := {||o, nCol| alert('bHeaderClick') }

        // Evento de duplo click na célula
        oBrowse:bLDb1Click := {|| alert('bLDb1Click') }

        // Cria Botoes com metodos básicos
        TButton():New( 160, 002, "GoUp()", oDlg,{|| oBrowse:GoUp(),
oBrowse:setFocus() },40,010,,,F...T...F...F...F...F... )
        TButton():New( 160, 052, "GoDown()", oDlg,{|| oBrowse:GoDown(),
oBrowse:setFocus() },40,010,,,F...T...F...F...F...F... )
        TButton():New( 160, 102, "GoTop()", oDlg,{||
oBrowse:GoTop(),oBrowse:setFocus() }, 40, 010,,,F...T...F...F...F...F...F...)
        TButton():New( 160, 152, "GoBottom()", oDlg,{||
oBrowse:GoBottom(),oBrowse:setFocus() },40,010,,,F...T...F...F...F...F...F...)
        TButton():New( 172, 002, "Linha atual", oDlg,{|| alert(oBrowse:nAt)
},40,010,,,F...T...F...F...F...F...F... )
        TButton():New( 172, 052, "Nr Linhas", oDlg,{|| alert(oBrowse:nLen)
},40,010,,,F...T...F...F...F...F...F... )
        TButton():New( 172, 102, "Linhas visiveis", oDlg,{||
alert(oBrowse:nRowCount()) },40,010,,,F...T...F...F...F...F...F... )
```

```
TButton():New( 172, 152, "Alias", oDlg,{|| alert(oBrowse:cAlias)
},40,010,,,,.F.,.T.,.F.,,.F.,,.F.)

ACTIVATE DIALOG oDlg CENTERED
RETURN
```

8. FWTemporaryTable

As tabelas temporárias são usadas com maior frequência para fornecer espaço de trabalho para os resultados intermediários no processamento de dados dentro de um lote ou de um procedimento. Classe para criação e manipulação de tabelas temporárias no BD

Sintaxe:

FWTemporaryTable():New([<cAlias>], [<aFields>])-> Objeto FWTemporaryTable

Nome	Tipo	Descrição	Default
cAlias	Caracter	Alias a ser utilizado pela tabela.	GetNextAlias
aFields	Array	Array com estrutura de campos: [1] Nome [2] Tipo [3] Tamanho [4] Decimal	{}

- FWTemporaryTable():Create(): Método responsável pela criação da tabela
- FWTemporaryTable():Delete(): Método responsável por efetuar a exclusão da tabela, e fechar o alias
- FWTemporaryTable():AddIndex(<cIndexName>, <aFields>): Adiciona um índice na tabela.
- FWTemporaryTable():GetRealName(): Retorna o nome com o qual a tabela foi criada no BD.
- FWTemporaryTable():InsertSelect(<cTableFrom>, <aFieldsFrom>): Efetua carga de todos os campos baseada em um select de outra tabela, utilizando o padrão INSERT INTO SELECT. Este método deve ser utilizado quando todos os campos da estrutura da tabela serão preenchidos.
- FWTemporaryTable():InsertIntoSelect(<aFieldsTo>, <cTableFrom>, <aFieldsFrom>): Efetua carga de todos os campos baseada em um select de outra tabela, utilizando o padrão INSERT INTO SELECT. Este método deve ser utilizado quando somente alguns campos da estrutura da tabela serão preenchidos.
- FWTemporaryTable():GetAlias(): Retorna o alias utilizado pelo arquivo.
- FWTemporaryTable():SetFields(<aFields>): Define os campos da estrutura.

Exemplo:

```

Local aFields := {}
Local oTempTable
Local nI
Local cAlias := "MEUALIAS"
Local cQuery

//-----
//Criação do objeto
//-----
oTempTable := FWTemporaryTable():New( cAlias )

//-----
//Monta os campos da tabela
//-----
aadd(aFields,{ "DESCR", "C", 30, 0})
aadd(aFields,{ "CONTR", "N", 3, 1})
aadd(aFields,{ "ALIAS", "C", 3, 0})

oTempTable:SetFields( aFields )
oTempTable:AddIndex("indice1", { "DESCR" } )
oTempTable:AddIndex("indice2", { "CONTR", "ALIAS" } )
//-----
//Criação da tabela
//-----
oTempTable:Create()

conout("Executando a cópia dos registros da tabela: " +
RetSqlName("CT0") )

//-----
-----
//Caso o INSERT INTO SELECT preencha todos os campos, este será um
método facilitador
//Caso contrário deverá ser chamado o InsertIntoSelect():
// oTempTable:InsertIntoSelect( { "DESCR", "CONTR" } ,
RetSqlName("CT0") , { "CT0_DESC", "CT0_CONTR" } )
//-----
-----
oTempTable:InsertSelect( RetSqlName("CT0") , { "CT0_DESC", "CT0_CONTR",
"CT0_ALIAS" } )

//-----
//Executa query para leitura da tabela
//-----
cQuery := "select * from " + oTempTable:GetRealName()
MP SysOpenQuery( cQuery, 'QRYTMP' )

DbSelectArea('QRYTMP')
    
```



```
while !eof()
  for nI := 1 to fcount()
    varinfo(fieldname(nI), fieldget(nI))
  next
  dbskip()
Enddo

//-----
//Exclui a tabela
//-----
oTempTable:Delete()

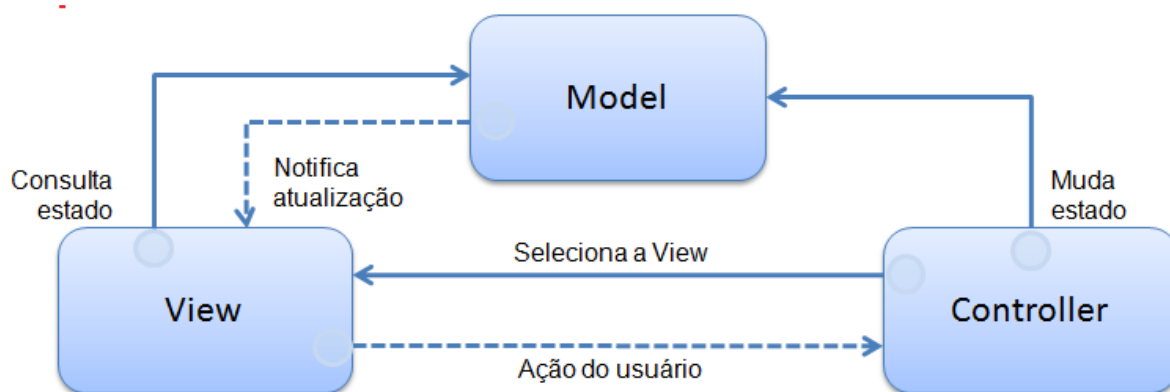
return
```

9. Arquitetura MVC

A arquitetura **Model-View-Controller** ou **MVC**, como é mais conhecida, é um padrão de arquitetura de software que visa separar a lógica de negócio da lógica de apresentação (a interface), permitindo o desenvolvimento, teste e manutenção isolados de ambos.

Aqueles que já desenvolveram uma aplicação em AdvPL vão perceber, que justamente a diferença mais importante entre a forma de construir uma aplicação em MVC e a forma tradicional é essa separação. E é ela que vai permitir o uso da regra de negócio em aplicações que tenham ou não interfaces, como Web Services e aplicação automática, bem como seu reuso em outras aplicações.

A arquitetura MVC possui três componentes básicos:



- **Model ou modelo de dados:** representa as informações do domínio do aplicativo e fornece funções para operar os dados, isto é, ele contém as funcionalidades do aplicativo. Nele definimos as regras de negócio: tabelas, campos, estruturas, relacionamentos etc. O modelo de dados (Model) também é responsável por notificar a interface (View) quando os dados forem alterados.

- **View ou interface:** responsável por renderizar o modelo de dados (Model) e possibilitar a interação do usuário, ou seja, é o responsável por exibir os dados.

- **Controller:** responde às ações dos usuários, possibilita mudanças no Modelo de dados (Model) e seleciona a View correspondente.

Para facilitar e agilizar o desenvolvimento, na implementação do MVC feita no AdvPL, o desenvolvedor trabalhará com as definições de Modelo de dados (Model) e View, a parte responsável pelo Controller já está intrínseca. Frisando bem, a grande mudança, o grande paradigma a ser quebrado na forma de pensar e se desenvolver uma aplicação em AdvPL utilizando MVC é a separação da regra de negócio da interface. Para que isso fosse possível foram desenvolvidas várias novas classes e métodos no AdvPL.

9.1. Principais funções da aplicação em AdvPL utilizando o MVC

Apresentamos agora o modelo de construção de uma aplicação em AdvPL utilizando o MVC. Os desenvolvedores em suas aplicações serão responsáveis por definir as seguintes funções:

- **ModelDef:** Contém a construção e a definição do Model, lembrando que o Modelo de dados (Model) contém as regras de negócio;
- **ViewDef:** Contém a construção e definição da View, ou seja, será a construção da interface;
- **MenuDef:** Contém a definição das operações disponíveis para o modelo de dados (Model).

Cada fonte em MVC (PRW) só pode conter uma de cada dessas funções. Só pode ter uma **ModelDef**, uma **ViewDef** e uma **MenuDef**. Ao se fazer uma aplicação em AdvPL utilizando MVC, automaticamente ao final, está aplicação já terá disponível.

- **Pontos de Entradas** já disponíveis;
- **Web Service** para sua utilização;
- **Importação ou exportação** mensagens XML.

9.2. O que é a função ModelDef?

A função ModelDef define a regra de negócios propriamente dita onde são definidas

- Todas as entidades (tabelas) que farão parte do modelo de dados (Model);
- Regras de dependência entre as entidades;
- Validações (de campos e aplicação);
- Persistência dos dados (gravação).

Para uma ModelDef não é preciso necessariamente possuir uma interface. Como a regra de negócios é totalmente separada da interface no MVC, podemos utilizar a ModelDef em qualquer outra aplicação, ou até utilizarmos uma determinada ModelDef como base para outra mais complexa.

As entidades da ModelDef não se baseiam necessariamente em metadados (dicionários). ela se baseia em estruturas e essas por sua vez é que podem vir do metadados ou serem construídas manualmente. A ModelDef deve ser uma Static Function dentro da aplicação.

9.3. O que é a função ViewDef?

A função ViewDef define como o será a interface e portanto como o usuário interage com o modelo de dados (Model) recebendo os dados informados pelo usuário, fornecendo ao modelo de dados (definido na ModelDef) e apresentando o resultado. A interface pode ser baseada totalmente ou parcialmente em um metadado (dicionário), permitindo:

- Reaproveitamento do código da interface, pois uma interface básica pode ser acrescida de novos componentes;
- Simplicidade no desenvolvimento de interfaces complexas. Um exemplo disso são aquelas aplicações onde uma GRID depende de outra. No MVC a construção de aplicações que tem GRIDs dependentes é extremamente fácil;
- Agilidade no desenvolvimento, a criação e a manutenção se tornam muito mais ágeis;
- Mais de uma interface por Business Object. Poderemos ter interfaces diferentes para cada variação de um segmento de mercado, como o varejo.

A **ViewDef** deve ser uma **Static Function** dentro da aplicação.

9.4. O que é a função MenuDef?

Uma função MenuDef define as operações que serão realizadas pela aplicação, tais como inclusão, alteração, exclusão, etc.

Deve retornar um *array* em um formato específico com as seguintes informações:

- Título
- Nome da aplicação associada
- Reservado;
- Tipo de Transação a ser efetuada.

E que podem ser:

- 1 para Pesquisar
- 2 para Visualizar
- 3 para Incluir
- 4 para Alterar
- 5 para Excluir
- 6 para Imprimir
- 7 para Copiar
- 5. Nível de acesso;
- 6. Habilita Menu Funcional;

10. Novo comportamento na interface

Nas aplicações desenvolvidas em AdvPL tradicional, após a conclusão de uma operação de alteração fecha-se a interface e retorna ao Browse.

Nas aplicações em MVC, após as operações de inclusão e alteração, a interface permanece ativa e no rodapé exibe-se a mensagem de que a operação foi bem sucedida.

10.1. Aplicações com Browsers (FWMBrowse)

Para a construção de uma aplicação que possui um Browse, o MVC utiliza a classe FWMBrowse. Esta classe exibe um objeto Browse que é construído a partir de metadados (dicionários). Esta classe não foi desenvolvida exclusivamente para o MVC, aplicações que não são em MVC também podem utilizá-la.

- Substituir componentes de Browse;
- Reduzir o tempo de manutenção, em caso de adição de um novo requisito;
- Ser independente do ambiente Microsiga Protheus.
- E apresenta como principais melhorias:
- Padronização de legenda de cores;
- Melhor usabilidade no tratamento de filtros;
- Padrão de cores, fontes e legenda definidas pelo usuário – Deficiente visual;
- Redução do número de operações no SGBD (no mínimo 3 vezes mais rápido);
- Novo padrão visual.

10.1.1. Construção básica de um Browse

Iniciamos a construção básica de um *Browse*. Primeiramente crie um objeto *Browse* da seguinte forma:

```
oBrowse := FWMBrowse():New()
```

Definimos a tabela que será exibida na *Browse* utilizando o método *SetAlias*. As colunas, ordens, etc. A exibição é obtida pelo metadados (dicionários).

```
oBrowse:SetAlias('SA1')
```

Definimos o título que será exibido como método *SetDescription*.

```
oBrowse:SetDescription('Cadastro de Cliente')
```

Necessário ativar a classe.

```
oBrowse:Activate()
```

Com esta estrutura básica construímos uma aplicação com *Browse*.

O *Browse* apresentado automaticamente já terá:

- Pesquisa de registro;
- Filtro configurável;
- Configuração de colunas e aparência;
- Impressão.

10.1.2. Legendas de um Browse (AddLegend)

Para o uso de legendas no *Browse* utilizamos o método *AddLegend*, que possui a seguinte sintaxe:

```
AddLegend( <cRegra>, <cCor>, <cDescrição> )
```

Exemplo:

```
oBrowse:AddLegend( "A1_TIPO == 'F'", "YELLOW", "Cons.Final" )
oBrowse:AddLegend( "ZA0_TIPO == 'L'", "BLUE", "Produtor Rural" )
```

cRegra: é a expressão em AdvPL para definir a legenda.

cCor: é o parâmetro que define a cor de cada item da legenda.

São possíveis os seguintes valores:

- **GREEN:** Para a cor Verde
- **RED:** Para a cor Vermelha
- **YELLOW:** Para a cor Amarela
- **ORANGE:** Para a cor Laranja
- **BLUE:** Para a cor Azul
- **GRAY:** Para a cor Cinza
- **BROWN:** Para a cor Marrom
- **BLACK:** Para a cor Preta
- **PINK:** Para a cor Rosa
- **WHITE:** Para a cor Branca

cDescrição: a que será exibida para cada item da legenda

Cada uma das legendas se tornará automaticamente uma opção de filtro. Cuidado ao montar as regras da legenda. Se houverem regras conflitantes será exibida a legenda correspondente à 1ª regra que for satisfeita.

10.1.3. Filtros de um Browse (SetFilterDefault)

Se quisermos definir um filtro para o Browse utilizamos o método SetFilterDefault, que possui a seguinte sintaxe:

```
SetFilterDefault ( <filtro> )
```

Exemplo:

```
oBrowse:SetFilterDefault ("A1_TIPO=='F'")  
ou  
oBrowse:SetFilterDefault ( "Empty(A1_ULTCOM) " )
```

A expressão de filtro é em AdvPL. O filtro definido na aplicação não anula a possibilidade do usuário fazer seus próprios filtros. Os filtros feitos pelo usuário serão aplicados em conjunto com o definido na aplicação (condição de AND).

O filtro da aplicação não poderá ser desabilitado pelo usuário.

10.1.4. Desabilitação de detalhes do Browse (DisableDetails)

Automaticamente para o Browse são exibidos, em detalhes, os dados da linha posicionada. Para desabilitar esta característica utilizamos o método **DisableDetails**.

Exemplo :

```
oBrowse:DisableDetails()
```

Normalmente, para se exibir campos virtuais nos Browsers, fazemos uso da função **Posicione**. No novo Browse esta prática se torna ainda mais importante, pois, quando ele encontra a função **Posicione** na definição de um campo virtual e a base de dados é um Sistema de Gerenciamento de Base de Dados - SGBD (usa o TOTVSDbAccess), o Browse acrescenta um INNER JOIN na query que será enviada ao SGBD, melhorando assim o desempenho para a extração dos dados. Portanto, sempre utilize a função **Posicione** para exibir campos virtuais.

10.2. Exemplo completo de Browse

```
User Function COMP011_MVC()

Local oBrowse
// Instanciamento da Classe de Browse
oBrowse := FWMBrowse():New()
// Definição da tabela do Browse
oBrowse:SetAlias('SA1')
// Definição da legenda
oBrowse:AddLegend( "A1_TIPO == 'F'", "YELLOW", "Cons.Final"      )
oBrowse:AddLegend( "ZA0_TIPO == 'L'", "BLUE", "Produtor Rural"  )
// Definição de filtro
oBrowse:SetFilterDefault( "A1_TIPO == 'F'" )
// Titulo da Browse
oBrowse:SetDescription('Cadastro de Clientes')

// Opcionalmente pode ser desligado a exibição dos detalhes
//oBrowse:DisableDetails()
// Ativação da Classe
oBrowse:Activate()

Return NIL
```

11. Construção de aplicação ADVPL utilizando MVC

Iniciamos agora a construção da parte em MVC da aplicação, que são as funções de ModeDef, que contém as regras de negócio e a ViewDef que contém a interface. Um ponto importante que deve ser observado é que, assim como a MenuDef, só pode haver uma função ModelDef e uma função ViewDef em uma fonte.

Se para uma determinada situação for preciso trabalhar em mais de um modelo de dados (Model), a aplicação deve ser quebrada em várias fontes (PRW) cada um com apenas uma ModelDef e uma ViewDef.

12. Criando o MenuDef

A criação da estrutura do **MenuDef** deve ser uma **Static Function** dentro da aplicação.

```
Static Function MenuDef()

Local aRotina := {}
aAdd( aRotina, { 'Visualizar', 'VIEWDEF.COMP021_MVC', 0, 2, 0, NIL } )
aAdd( aRotina, { 'Incluir', 'VIEWDEF.COMP021_MVC', 0, 3, 0, NIL } )
aAdd( aRotina, { 'Alterar', 'VIEWDEF.COMP021_MVC', 0, 4, 0, NIL } )
aAdd( aRotina, { 'Excluir', 'VIEWDEF.COMP021_MVC', 0, 5, 0, NIL } )
aAdd( aRotina, { 'Imprimir', 'VIEWDEF.COMP021_MVC', 0, 8, 0, NIL } )
aAdd( aRotina, { 'Copiar', 'VIEWDEF.COMP021_MVC', 0, 9, 0, NIL } )

Return aRotina
```


Sempre referenciaremos a ViewDef de um fonte, pois ela é a função responsável pela a interface da aplicação. Para facilitar o desenvolvimento, no MVC a MenuDef escreva-a da seguinte forma:

```
Static Function MenuDef()
Local aRotina := {}

ADD OPTION aRotina Title 'Visualizar' Action 'VIEWDEF.COMP021_MVC' OPERATION 2
ACCESS 0
ADD OPTION aRotina Title 'Incluir'      Action 'VIEWDEF.COMP021_MVC' OPERATION 3
ACCESS 0
ADD OPTION aRotina Title 'Alterar'      Action 'VIEWDEF.COMP021_MVC' OPERATION 4
ACCESS 0
ADD OPTION aRotina Title 'Excluir'      Action 'VIEWDEF.COMP021_MVC' OPERATION 5
ACCESS 0
ADD OPTION aRotina Title 'Imprimir'     Action 'VIEWDEF.COMP021_MVC' OPERATION 8
ACCESS 0
ADD OPTION aRotina Title 'Copiar'       Action 'VIEWDEF.COMP021_MVC' OPERATION 9
ACCESS 0

Return aRotina
```

- **TITLE:** nome do item no menu
- **ACTION:** 'VIEWDEF.nome_do_arquivo_fonte' PRW
- **OPERATION:** Valor que define a operação a ser executada(inclusão,alteração,etc)
- **ACCESS:** Valor que define o nível de acesso

Podemos criar um menu com opções padrão para o MVC utilizando a função **FWMVCMENU**

```
Static Function MenuDef()
Return FWMVCMenu( 'COMP011_MVC' ) )
```

Será criado um menu padrão com as opções: **Visualizar, Incluir, Alterar, Excluir, Imprimir e Copiar.**

13. Construção da função ModelDef

Nessa função são definidas as regras de negócio ou modelo de dados (*Model*). Elas contêm as definições de:

- Entidades envolvidas;
- Validações;
- Relacionamentos;

- Persistência de dados (gravação);

Iniciamos a função **ModelDef**:

```
Static Function ModelDef()  
Local oModel // Modelo de dados que será construído
```

Construindo o Model

```
oModel := MPFormModel():New( 'COMP011M' )
```

MPFormModel é a classe utilizada para a construção de um objeto de modelo de dados (**Model**).

Devemos dar um identificador (**ID**) para o modelo como um todo e também um para cada componente. Essa é uma característica do MVC, todo componente do modelo ou da *interface* devem ter um ID, como formulários, GRIDs, boxes, etc.

COMP011M é o identificador (**ID**) dado ao Model, é importante ressaltar com relação ao identificador (**ID**) do Model:

Se a aplicação é uma *Function*, o identificador (**ID**) do modelo de dados (**Model**) não pode ter o mesmo nome da função principal e esta prática é recomendada para facilitar a codificação. Por exemplo, se estamos escrevendo a função XPTO, o identificador (**ID**) do modelo de dados (**Model**) não poderá ser XPTO.

13.1. Construção de uma estrutura de dados (FWFormStruct)

A primeira coisa que precisamos fazer é criar a estrutura utilizada no modelo de dados (Model). As estruturas são objetos que contêm as definições dos dados necessárias para uso da ModelDef ou para a ViewDef.

- Estrutura dos Campos;
- Índices;
- Gatilhos;
- Regras de preenchimento;

O MVC não trabalha vinculado aos metadados (dicionários) do Microsiga Protheus, ele trabalha vinculado a estruturas. Essas estruturas, por sua vez, é que podem ser construídas a partir dos metadados. Com a função FWFormStruct a estrutura será criada a partir do metadado.

Sua sintaxe:

FWFormStruct(<nTipo>, <cAlias>)

- **nTipo**: Tipo da construção da estrutura: 1 para Modelo de dados (Model) e 2 para interface (View);
- **cAlias**: Alias da tabela no metadado;

Exemplo:

```
Local oStruSA1 := FWFormStruct( 1, 'SA1' )
```

No exemplo, o objeto oStruSA1 será uma estrutura para uso em um modelo de dados (**Model**). O primeiro parâmetro (1) indica que a estrutura é para uso no modelo e o segundo parâmetro indica qual a tabela dos metadados será usada para a criação da estrutura (SA1).

Para modelo de dados (**Model**), a função FWFormStruct, traz para a estrutura todos os campos que compõem a tabela independentemente do nível, uso ou módulo. Considera também os campos virtuais.
Para a interface (**View**) a função **FWFormStruct**, traz para a estrutura os campos conforme o nível, uso ou módulo.

13.2. Criação de componente no modelo de dados (AddFields)

O método **AddFields** adiciona um componente de formulário ao modelo.

A estrutura do modelo de dados (Model) deve iniciar, obrigatoriamente, com um componente de formulário.

Exemplo:

```
oModel:AddFields( 'SA1MASTER', /*cOwner*/, oStruSA1 )
```

Devemos dar um identificador (ID) para cada componente do modelo.

- **SA1MASTER**: é o identificador (ID) dado ao componente de formulário no modelo, **oStruZA0** é a estrutura que será usada no formulário e que foi construída anteriormente utilizando **FWFormStruct**, note que o segundo parâmetro (*owner*) não foi informado, isso porque este é o 1º componente do modelo, é o **Pai** do modelo de dados (*Model*) e portanto não tem um componente superior ou *owner*.

13.3. Criação de um componente na interface (AddField)

Adicionamos na interface (View) um controle do tipo formulário (antiga *enchoice*), para isso usamos o método **AddField**. A interface (View) deve iniciar, obrigatoriamente, com um componente do tipo formulário.

```
oView:AddField( 'VIEW_SA1', oStruSA1, 'SA1MASTER' )
```

Devemos dar um identificador (ID) para cada componente da interface (View).

VIEW_SA1 é o identificador (ID) dado ao componente da interface (View), **oStruSA1** é a estrutura que será usada e **SA1MASTER** é identificador (ID) do componente do modelo de dados (Model) vinculado a este componente da interface (View).

Cada componente da interface (View) deve ter um componente do modelo de dados (Model) relacionado, isso equivale a dizer que os dados do **SA1MASTER** serão exibidos na interface (View) no componente **VIEW_SA1**

13.4. Descrição dos componentes do modelo de dados (SetDescription)

Sempre definindo uma descrição para os componentes do modelo. Com o método **SetDescription** adicionamos a descrição ao modelo de dados (Model), essa descrição será usada em vários lugares como em *Web Services* por exemplo.

Adicionamos a descrição do modelo de dados:

```
oModel:SetDescription( 'Modelo de dados Cliente')
```

Adicionamos a descrição dos componentes do modelo de dados:

```
oModel:GetModel( 'SA1MASTER' ):SetDescription( 'Dados dos Clientes' )
```

Para um modelo que só contém um componente parece ser redundante darmos uma descrição para o modelo de dados (Model) como um todo e uma para o componente, mas quando estudarmos outros modelos onde haverá mais de um componente esta ação ficará mais clara.

13.5. Finalização de ModelDef

Ao final da função **ModelDef**, deve ser retornado o objeto de modelo de dados (Model) gerado na função.

```
Return oModel
```

Exemplo completo da ModelDef

```
Static Function ModelDef()  
  
Local oModel // Modelo de dados que será construído  
  
// Cria o objeto do Modelo de Dados  
oModel := MPFormModel():New('COMP011M' )  
  
// Cria a estrutura a ser usada no Modelo de Dados  
Local oStruSA1 := FWFormStruct( 1, 'SA1' )  
  
// Adiciona ao modelo um componente de formulário  
oModel:AddFields( 'SA1MASTER', /*cOwner*/, oStruSA1)  
  
// Adiciona a descrição do Modelo de Dados  
oModel:SetDescription( 'Modelo de dados de Clientes' )  
  
// Adiciona a descrição do Componente do Modelo de Dados  
oModel:GetModel( 'SA1MASTER' ):SetDescription( 'Dados do Cliente' )
```

```
// Retorna o Modelo de dados  
Return oModel
```

14. Construção da função ViewDef

A interface (View) é responsável por renderizar o modelo de dados (Model) e possibilitar a interação do usuário, ou seja, é o responsável por exibir os dados.

O **ViewDef** contém a definição de toda a parte visual da aplicação.

Iniciamos a função:

```
Static Function ViewDef()
```

A interface (View) sempre trabalha baseada em um modelo de dados (Model). Criaremos um objeto de modelo de dados baseado no **ModelDef** que desejamos.

Com a função **FWLoadModel** obtemos o modelo de dados (Model) que está definido em um fonte, no nosso caso é o próprio fonte, mas nada impediria o acesso do modelo de qualquer outro fonte em MVC, com isso podemos reaproveitar o modelo de dados (Model) em mais de uma interface (View).

```
Local oModel := FWLoadModel('COMP011M')
```

- **COMP011_MVC**: é nome do fonte de onde queremos obter o modelo de dados (Model).

Iniciando a construção da interface (View).

```
oView := FWFormView():New()
```

- **FWFormView**: é a classe que deverá ser usada para a construção de um objeto de interface (View).

Definimos qual o modelo de dados (Model) que será utilizado na interface (View).

```
oView:SetModel( oModel )
```

14.1. Exibição dos dados (CreateHorizontalBox / CreateVerticalBox)

Sempre precisamos criar um contêiner, um objeto, para receber algum elemento da interface (View). Em MVC criaremos sempre box horizontal ou vertical para isso.

O método para criação de um box horizontal é:

```
oView>CreateHorizontalBox( 'TELA' , 100 )
```

Devemos dar um identificador (*ID*) para cada componente da *interface* (*View*).

- **TELA:** é o identificador (ID) dado ao **box** e o número **100** representa o percentual da tela que será utilizado pelo **Box**.

No MVC não há referências a coordenadas absolutas de tela, os componentes visuais são sempre **All Client**, ou seja, ocuparão todo o contêiner onde for inserido

14.2. Relacionando o componente da interface (SetOwnerView)

Precisamos relacionar o componente da interface (*View*) com um **box** para exibição, para isso usamos o método **SetOwnerView**.

```
oView:SetOwnerView( 'VIEW_SA1', 'TELA' )
```

Desta forma o componente *VIEW_SA1* será exibido na tela utilizando o box *TELA*.

14.3. Finalização da ViewDef

Ao final da função *ViewDef*, deve ser retornado o objeto de interface (*View*) gerado

```
Return oView
```

Exemplo completo da *ViewDef*

```
Static Function ViewDef()

// Cria um objeto de Modelo de dados baseado no ModelDef() do fonte informado
Local oModel := FWLoadModel( 'COMP011_MVC' )

// Cria a estrutura a ser usada na View
Local oStruZA0 := FWFormStruct( 2, 'SA1' )

// Interface de visualização construída
Local oView

// Cria o objeto de View
oView := FWFormView():New()

// Define qual o Modelo de dados será utilizado na View
oView:SetModel( oModel )

// Adiciona no nosso View um controle do tipo formulário
// (antiga Enchoice)
oView:AddField( 'VIEW_SA1', oStruSA1, 'SA1MASTER' )
```

```
// Criar um "box" horizontal para receber algum elemento da view
oView:CreateHorizontalBox( 'TELA' , 100 )

// Relaciona o identificador (ID) da View com o "box" para exibição
oView:SetOwnerView( 'VIEW_SA1', 'TELA' )

// Retorna o objeto de View criado
Return oView
```

15. Objetos de interface

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

RWMAKE.CH: permite a utilização da sintaxe CLIPPER na definição dos componentes visuais.

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

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

Exemplo 01 - Include Rwmake.ch

```
#include "rwmake.ch"

@ 0,0 TO 400,600 DIALOG oDlg TITLE "Janela em sintaxe Clipper" ;
ACTIVATE DIALOG oDlg CENTERED
```

Exemplo 02 - Include Protheus.ch

```
#include "protheus.ch"

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

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

- ☐ **BUTTON()**
- ☐ **CHECKBOX()**
- ☐ **COMBOBOX()**
- ☐ **FOLDER()**
- ☐ **MSDIALOG()**
- ☐ **MSGET()**
- ☐ **RADIO()**

 SAY()

 SBUTTON()

BUTTON()

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

CHECKBOX()

Sintaxe	@ nLinha,nColuna CHECKBOX oCheckBox VAR VARIABEL PROMPT cTexto WHEN WHEN UNIDADE OF oObjetoRef SIZE nLargura,nAltura MESSAGE cMensagem
Descrição	Define o componente visual ComboBox, o qual permite seleção de um item dentro de uma lista de opções de textos simples no formato de um vetor.

FOLDER()

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

MSDIALOG()

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

MSGET()

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

RADIO()

Sintaxe	@ nLinha,nColuna RADIO oRadio VAR nRadio 3D SIZE nLargura,nAltura <ITEMS
----------------	--

	PROMPT> cItem1,cItem2,...,cItemX OF oObjetoRef UNIDADE ON CHANGE CHANGE ON CLICK CLICK
Descrição	Define o componente visual Radio, também conhecido como RadioMenu, o qual é seleção de uma opção ou de múltiplas opções através de uma marca para os itens exibidos de uma lista. Difere do componente CheckBox, pois cada elemento de check é sempre único, e o Radio pode conter um ou mais elementos.

SAY()






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

SBUTTON()

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

15.1. Réguas de processamento

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

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

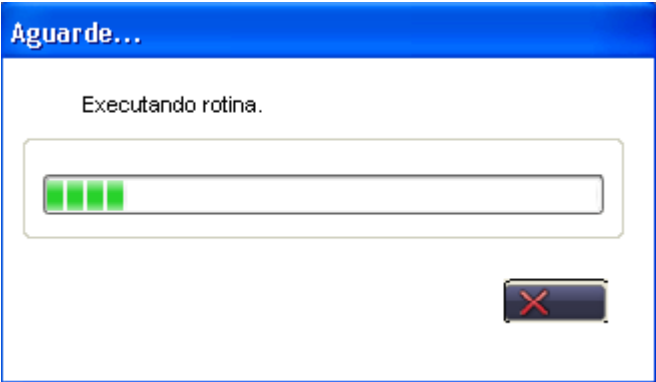
15.1.1. RptStatus()

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

Sintaxe: RptStatus(bAcao, cMensagem)

Retorno: Nil

Parâmetros:

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

Exemplo: Função RPTStatus() e acessórias

```
User Function GRptStatus()  
Local aSay      := {}  
Local aButton   := {}  
Local nOpc      := 0  
Local cTitulo   := "Exemplo de Funções"  
Local cDesc1    := "Este programa exemplifica a utilização da função;  
  
Processa() em conjunto"  
Local cDesc2    := "com as funções de incremento ProcRegua() e IncProc()"  
Private cPerg   := "RPTSTA"  
  
CriaSX1()  
Pergunte(cPerg,.F.)  
  
AADD( aSay, cDesc1 )  
AADD( aSay, cDesc2 )  
  
AADD( aButton, { 5, .T., {|| Pergunte(cPerg,.T. )    } } )  
AADD( aButton, { 1, .T., {|| nOpc := 1, FechaBatch() } } )  
AADD( aButton, { 2, .T., {|| FechaBatch()            } } )  
  
FormBatch( cTitulo, aSay, aButton )  
  
If nOpc <> 1  
    Return Nil  
Endif  
  
RptStatus({|lEnd|RunProc(@lEnd)}, "Aguarde...", "Executando rotina.", .T. )
```

Return Nil

Exemplo: Funções acessórias da RPTStatus()

```
Static Function RunProc(lEnd)
Local nCnt := 0

dbSelectArea("SX5")
dbSetOrder(1)
dbSeek(xFilial("SX5")+mv_par01,.T.)

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

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

SetRegua(nCnt)

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

Return .T.
```

SETREGUA()

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

Sintaxe: SetRegua(nMaxProc)

Parâmetros:

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

Retorno:

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

Exemplo:

```
dbSelectArea("SA1")
dbGoTop()
SetRegua>LastRec())
While !Eof()
    IncRegua()
    If Li > 60
...

```

INCREGUA()

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

Sintaxe: IncRegua(cMensagem)

Parâmetros:

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

Retorno:

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

Exemplo:

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

```

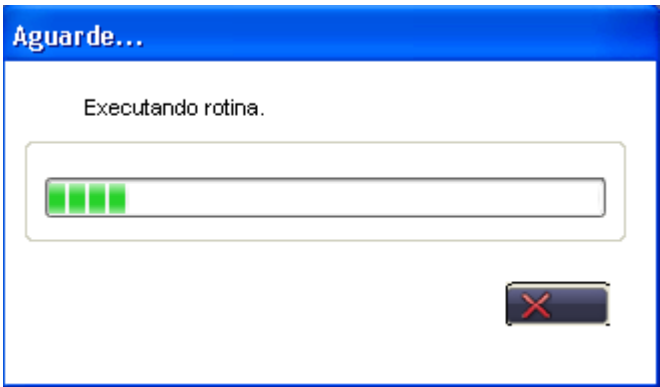
15.1.2. Processa()

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

Sintaxe: Processa(bAcao, cMensagem)

Retorno: Nil

Parâmetros:

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

Exemplo: Função PROCessa() e acessórias

```

User Function GProces1()
Local aSay      := {}
Local aButton  := {}
Local nOpc     := 0
Local cTitulo  := "Exemplo de Funções"
Local cDesc1   := "Utilização da função Processa()"
Local cDesc2   := " em conjunto com as funções de incremento ProcRegua() e"
Local cDesc3   := " IncProc()"

Private cPerg := "PROCES"

CriaSX1()
Pergunte(cPerg,.F.)

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

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

FormBatch( cTitulo, aSay, aButton )

If nOpc <> 1
    Return Nil
Endif

Processa( {||lEnd|RunProc(@lEnd)}, "Aguarde...", "Executando rotina.", .T. )

Return Nil
    
```

```

Static Function RunProc(lEnd)
Local nCnt := 0

dbSelectArea("SX5")
dbSetOrder(1)
dbSeek(xFilial("SX5")+mv_par01,.T.)

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

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

ProcRegua(nCnt)
While !Eof().And. X5_FILIAL == xFilial("SX5").And. X5_TABELA <= mv_par02
  IncProc("Processando tabela: "+SX5->X5_CHAVE)
  If lEnd
    MsgInfo(cCancela,"Fim")
    Exit
  Endif
  dbSkip()
End
Return .T.

```

SETPROC()

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

Sintaxe: Processa(nMaxProc)

Parâmetros:

nMaxProc

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

Retorno:

Nenhum

.

Exemplo:

```

...
dbSelectArea("SA1")
dbGoTop()
SetProc>LastRec())
While !Eof()
  IncProc()
  If Li > 60
...

```

INCPROC()

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

Sintaxe: IncProc(cMensagem)

Parâmetros:

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

Retorno:

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

Exemplo:

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

```

15.1.3. MsNewProcess().

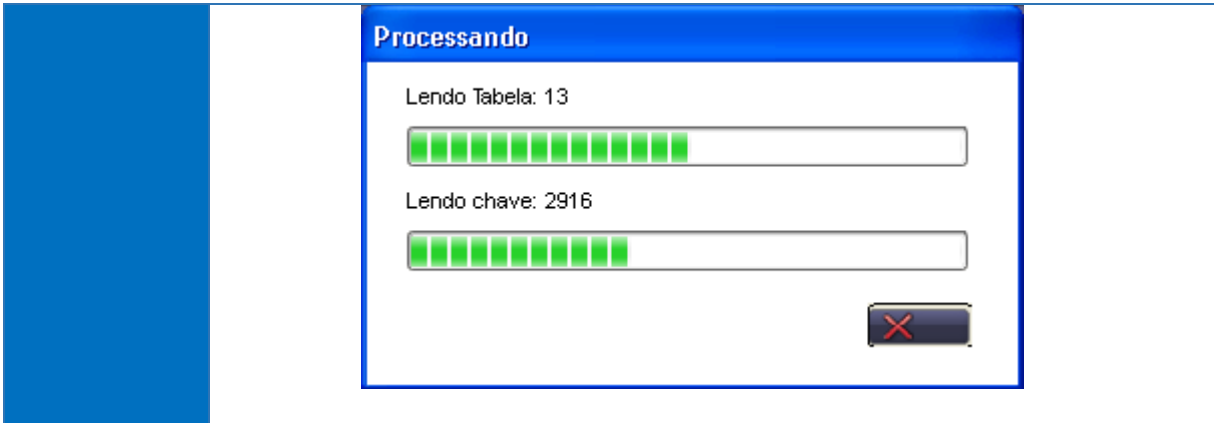
Régua de processamento dupla, possuindo dois indicadores de progresso independentes, utilizada no processamento de rotinas.

Sintaxe: MsNewProcess():New(bAcao, cMensagem)

Retorno: oProcess → objeto do tipo MsNewProcess()

Parâmetros:

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



Métodos:

Activate()	Inicia a execução do objeto MsNewProcess instanciado.
SetRegua1()	Define a quantidade de informações que serão demonstradas pelo indicador de progresso superior. Parâmetro: nMaxProc
IncRegua1()	Incrementa em uma unidade o indicador de progresso superior, o qual irá demonstrar a evolução do processamento de acordo com a quantidade definida pelo método SetRegua1(). Parâmetro: cMensagem
SetRegua2()	Define a quantidade de informações que serão demonstradas pelo indicador de progresso inferior. Parâmetro: nMaxProc
IncRegua2()	Incrementa em uma unidade o indicador de progresso inferior, o qual irá demonstrar a evolução do processamento de acordo com a quantidade definida pelo método SetRegua2(). Parâmetro: cMensagem

Exemplo: Objeto MsNewProcess() e métodos acessórios

```
User Function GProces2()
Private oProcess := NIL

oProcess := MsNewProcess():New({|lEnd| RunProc(lEnd,oProcess)};
"Processando","Lendo...",.T.)
oProcess:Activate()

Return Nil

Static Function RunProc(lEnd,oObj)
Local i := 0
Local aTabela := {}
Local nCnt := 0

aTabela := {"00",0}, {"13",0}, {"35",0}, {"T3",0}

dbSelectArea("SX5")
cFilialSX5 := xFilial("SX5")
```

```

dbSetOrder(1)
For i:=1 To Len(aTabela)
    dbSeek(cFilialSX5+aTabela[i,1])
    While !Eof() .And. X5_FILIAL+X5_TABELA == cFilialSX5+aTabela[i,1]
        If lEnd
            Exit
        Endif
        nCnt++
        dbSkip()
    End
    aTabela[i,2] := nCnt
    nCnt := 0
Next i

oObj:SetRegua1(Len(aTabela))
For i:=1 To Len(aTabela)
    If lEnd
        Exit
    Endif
    oObj:IncRegua1("Lendo Tabela: "+aTabela[i,1])
    dbSelectArea("SX5")
    dbSeek(cFilialSX5+aTabela[i,1])
    oObj:SetRegua2(aTabela[i,2])
    While !Eof() .And. X5_FILIAL+X5_TABELA == cFilialSX5+aTabela[i,1]
        oObj:IncRegua2("Lendo chave: "+X5_CHAVE)
        If lEnd
            Exit
        Endif
        dbSkip()
    End
Next i
Return
    
```

15.1.4. MsAguarde().

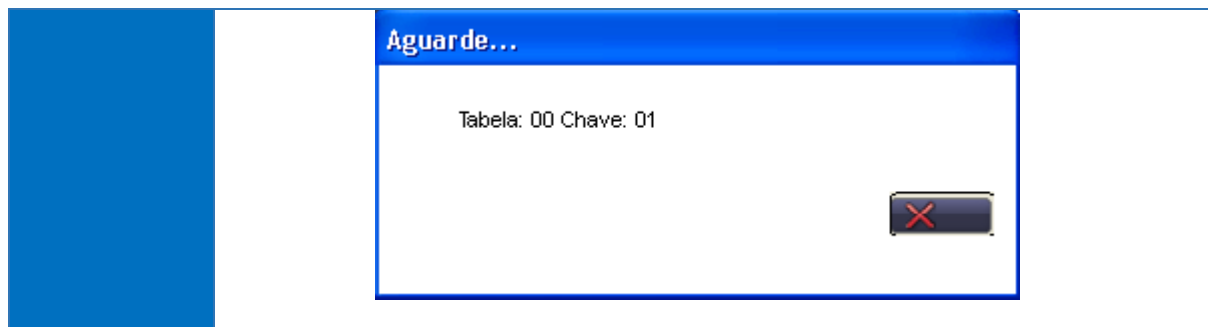
Indicador de processamento sem incremento

Sintaxe: Processa(bAcao, cMensagem, cTitulo)

Retorno: Nil

Parâmetros:

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



Exemplo: MSAguarde()

```

USER FUNCTION GMSAguarde()

PRIVATE lEnd := .F.
MSAguarde({|lEnd| RunProc(@lEnd)}, "Aguarde...", "Processando Clientes", .T.)
RETURN

STATIC FUNCTION RunProc(lEnd)

dbSelectArea("SX5")
dbSetOrder(1)
dbGoTop()

While !Eof()
    If lEnd
        MsgInfo(cCancel, "Fim")
        Exit
    Endif
    MsProcTxt("Tabela: "+SX5->X5_TABELA+" Chave: "+SX5->X5_CHAVE)
    dbSkip()
End

RETURN

```


15.1.5. MsgRun().

Indicador de processamento sem incremento.

Sintaxe: Processa(cMensagem, cTitulo, bAcao)

Retorno: Nil

Parâmetros:

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

Exemplo: MSGRun()

```

USER FUNCTION GMsgRun()

LOCAL nCnt := 0

dbSelectArea("SX1")
dbGoTop()

MsgRun("Lendo arquivo, aguarde...", "Título opcional", ;
      {||dbEval({|x| nCnt++}) })

MsgInfo("Ufa!!!, li "+AllTrim(Str(nCnt))+" registros", FunName())

RETURN()
    
```

15.2. ListBox()

A sintaxe clássica da linguagem ADVPL permite que o componente visual ListBox implemente dois tipos distintos de objetos:

- **Lista simples:** lista de apenas uma coluna no formato de um vetor, a qual não necessita da especificação de um cabeçalho.
- **Lista com colunas:** lista com diversas colunas que necessita de um cabeçalho no formato de um aHeader (array de cabeçalho).

15.2.1. ListBox simples

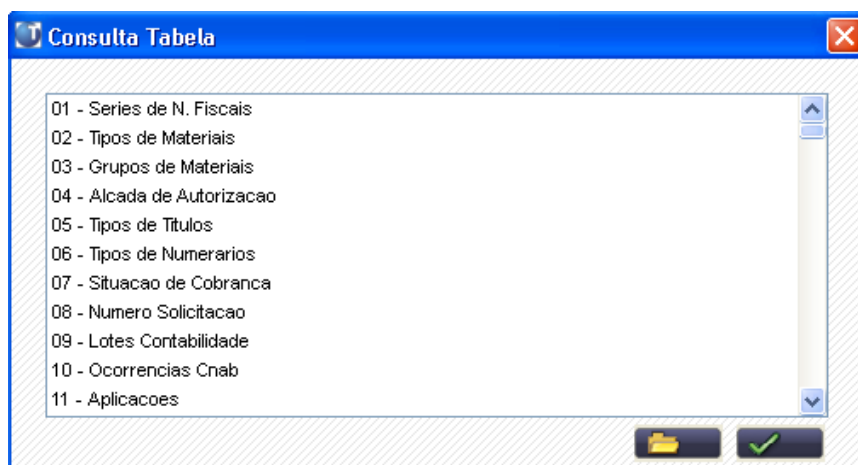
Sintaxe:

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

Parâmetros:

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

Aparência:



Exemplo: LISTBOX como lista simples

```
User Function ListBoxIte()

Local aVetor      := {}
Local oDlg        := Nil
Local oLbx        := Nil
Local cTitulo     := "Consulta Tabela"
Local nChave      := 0
Local cChave      := ""

dbSelectArea("SX5")
dbSetOrder(1)
dbSeek(xFilial("SX5"))

CursorWait()
```

```
//+-----+
//| Carrega o vetor conforme a condição |
//+-----+

While !Eof() .And. X5_FILIAL == xFilial("SX5") .And. X5_TABELA=="00"
    AADD( aVetor, Trim(X5_CHAVE)+" - "+Capital(Trim(X5_DESCRI)) )
    dbSkip()
EndDo

CursorArrow()

If Len( aVetor ) == 0
    Aviso( cTitulo, "Não existe dados a consultar", {"Ok"} )
    Return
Endif

//+-----+
//| Monta a tela para usuário visualizar consulta |
//+-----+
DEFINE MSDIALOG oDlg TITLE cTitulo FROM 0,0 TO 240,500 PIXEL

@ 10,10 LISTBOX oLbx VAR nChave ITEMS aVetor SIZE 230,95 OF oDlg PIXEL

oLbx:bChange := {|| cChave := SubStr(aVetor[nChave],1,2) }

DEFINE SBUTTON FROM 107,183 TYPE 14 ACTION LoadTable(cChave) ;
    ENABLE OF oDlg
DEFINE SBUTTON FROM 107,213 TYPE 1 ACTION oDlg:End() ;
    ENABLE OF oDlg

ACTIVATE MSDIALOG oDlg CENTER

Return
```

Exemplo: LISTBOX como lista simples – funções acessórias

```
STATIC FUNCTION LoadTable(cTabela)

LOCAL aTabela := {}
LOCAL oDlg := NIL
LOCAL oLbx := NIL

dbSelectArea("SX5")
dbSeek(xFilial("SX5")+cTabela)

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

If Len(aTabela)==0
    Aviso( "FIM", "Necessário selecionar um item", {"Ok"} )
    Return
Endif
```

```

DEFINE MSDIALOG oDlg TITLE "Dados da tabela selecionada" FROM 300,400
TO 540,900 PIXEL
@ 10,10 LISTBOX oLbx FIELDS HEADER "Tabela", "Descrição" SIZE 230,095
OF oDlg PIXEL
oLbx:SetArray( aTabela )
oLbx:bLine := {|| {aTabela[oLbx:nAt,1],aTabela[oLbx:nAt,2]} }
DEFINE SBUTTON FROM 107,213 TYPE 1 ACTION oDlg:End() ENABLE OF oDlg
ACTIVATE MSDIALOG oDlg

RETURN

```

15.2.2. ListBox múltiplas colunas

Sintaxe:

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

Parâmetros:

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

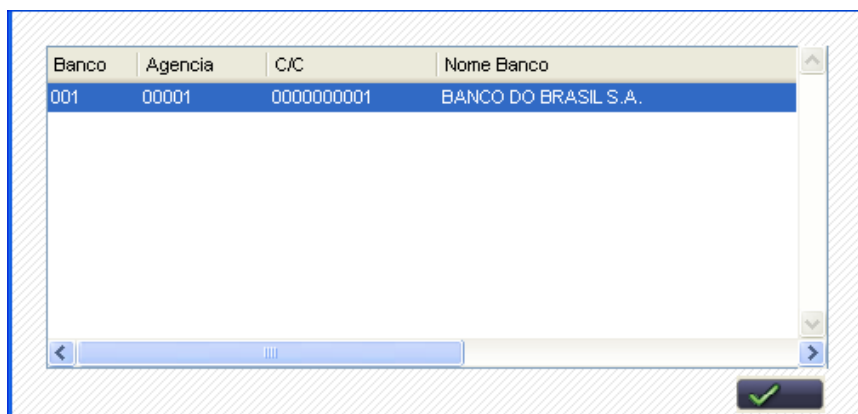
Metodos:

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

Atributos:

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

Aparência:



Exemplo: LISTBOX com grid

```
#include "protheus.ch"

User Function List_Box()

Local aVetor := {}
Local oDlg
Local oLbx
Local cTitulo := "Cadastro de Bancos"
Local cFilSA6

dbSelectArea("SA6")
dbSetOrder(1)
cFilSA6 := xFilial("SA6")
dbSeek(cFilSA6)

// Carrega o vetor conforme a condição.
While !Eof() .And. A6_FILIAL == cFilSA6
    AADD( aVetor, ;
        {A6_COD,A6_AGENCIA,A6_NUMCON,A6_NOME,A6_NREDUZ,A6_BAIRRO,A6_MUN})
    dbSkip()
End

// Se não houver dados no vetor, avisar usuário e abandonar rotina.
If Len( aVetor ) == 0
    Aviso( cTitulo, "Não existe dados a consultar", {"Ok"} )
    Return
Endif

// Monta a tela para usuário visualizar consulta.
DEFINE MSDIALOG oDlg TITLE cTitulo FROM 0,0 TO 240,500 PIXEL
    // Primeira opção para montar o listbox.
```

```
@ 10,10 LISTBOX oLbx FIELDS HEADER ;
"Banco", "Agencia", "C/C", "Nome Banco", "Fantasia", "Bairro", "Município" ;
SIZE 230,95 OF oDlg PIXEL

oLbx:SetArray( aVetor )
oLbx:bLine := {|| {aVetor[oLbx:nAt,1],,
                  aVetor[oLbx:nAt,2],,
                  aVetor[oLbx:nAt,3],,
                  aVetor[oLbx:nAt,4],,
                  aVetor[oLbx:nAt,5],,
                  aVetor[oLbx:nAt,6],,
                  aVetor[oLbx:nAt,7]}}

// Segunda opção para monta o listBox

oLbx := TWBrowse():New(10,10,230,95,,aCabecalho,, oDlg,,,,,,,,,,,,.F.;
,,,T,,,,F,,,,)
oLbx:SetArray( aVetor )
oLbx:bLine := {||aEval(aVetor[oLbx:nAt],{|z,w| aVetor[oLbx:nAt,w] } ) }
DEFINE SBUTTON FROM 107,213 TYPE 1 ACTION oDlg:End() ENABLE OF oDlg

ACTIVATE MSDIALOG oDlg CENTER

Return
```

15.3. ScrollBox()

O ScrollBox é o objeto utilizado para permitir a um Dialog exibir barras de rolagem verticais e Horizontais. Algumas aplicações com objetos definem automaticamente o ScrollBox, tais como:

- **Enchoice() ou MsMGet()**
- **NewGetDados()**
- **ListBox()**

Quando é definido um objeto ScrollBox, os demais componentes da janela deverão referenciar este objeto e não mais o objeto Dialog.

Desta forma o ScrollBox é atribuído a um objeto Dialog, e os componentes ao ScrollBox.

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

Sintaxe:

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

Parâmetros:

nLinha,nColuna	Posição do objeto ListBox em função da janela em que ele será definido. Posição do objeto ScrollBox em função da janela em que ele será definido.
oScrollBox	Objeto ScrollBox que será criado.

HORIZONTAL	Parâmetro que quando definido habilita a régua de rolagem horizontal.
VERTICAL	Parâmetro que quando definido habilita a régua de rolagem vertical.
nLargura,nAltura	Dimensões do objeto ScrollBox.
oObjetoRef	Objeto dialog no qual o componente será definido.
BORDER	Parâmetro que quando definido habilita a exibição de uma borda de delimitação do ScrollBox em relação a outros objetos.

Exemplo: Utilização de múltiplos ScrollBoxes

```
#INCLUDE "PROTHEUS.CH"

USER FUNCTION Scroll()

LOCAL oDlg := NIL
LOCAL oScroll := NIL
LOCAL oLbx1 := NIL
LOCAL oLbx2 := NIL
LOCAL bGet := NIL
LOCAL oGet := NIL
LOCAL aAIIPM := {}
LOCAL aTitulo := {}
LOCAL nTop := 5
LOCAL nWidth := 0
LOCAL cGet := ""
LOCAL cPict := ""
LOCAL cVar := ""
LOCAL n := 0

PRIVATE cTitulo := "Consulta Parcelamento"
PRIVATE aSay := {}
PRIVATE cProcesso,cPrefixo,cTipo,cCliente,cLoja,cNome,cCGC
PRIVATE dData,nTotal,nUFESP,cStatus,cCond

cProcesso := "P00001"
cPrefixo := "UNI"
cTipo := "MAN"
cCliente := "000001"
cLoja := "01"
cNome := "JOSE DA SILVA SANTOS SOARES"
cCGC := "00.000.000/0001-91"
dData := "26/03/03"
nTotal := 5922.00
nUFESP := 1000.00
cStatus := "Z"
cCond := "001"

// Vetor para os campos no Scrooll Box
//+-----+
//| aSay[n][1] - Titulo |
//| aSay[n][2] - Tipo |
//| aSay[n][3] - Tamanho |
//| aSay[n][4] - Decimal |
//| aSay[n][5] - Conteúdo/Variável |
```



```

//| aSay[n][6] - Formato |
//+-----+
AADD(aSay,{"Processo"      ,"C",06,0,"cProcesso" ,"@"})
AADD(aSay,{"Prefixo"      ,"C",03,0,"cPrefixo"  ,"@"})
AADD(aSay,{"Tipo"         ,"C",03,0,"cTipo"     ,"@"})
AADD(aSay,{"Cliente"      ,"C",06,0,"cCliente"  ,"@"})
AADD(aSay,{"Loja"         ,"C",02,0,"cLoja"     ,"@"})
AADD(aSay,{"Nome"         ,"C",30,0,"cNome"     ,"@"})
AADD(aSay,{"CNPJ/CPF"     ,"C",14,0,"cCGC"      ,"@"})
AADD(aSay,{"Dt.Processo"  ,"D",08,0,"dData"     ,"@"})
AADD(aSay,{"Total R$"     ,"N",17,2,"nTotal"    ,"@"})
AADD(aSay,{"Total UFESP"  ,"N",17,2,"nUFESP"    ,"@"})
AADD(aSay,{"Status"      ,"C",01,0,"cStatus"   ,"@"})
AADD(aSay,{"Cond.Pagto"   ,"C",03,0,"cCond"    ,"@"})

// Vetor para List Box
AADD(aAIIPM,{"1234","DCD9815","26/03/03"})
AADD(aAIIPM,{"1234","DCD9815","26/03/03"})
AADD(aAIIPM,{"1234","DCD9815","26/03/03"})

// Vetor para List Box
AADD(aTitulo,{"A","26/03/03","26/03/03","1.974,00","100,00"})
AADD(aTitulo,{"A","26/03/03","26/03/03","1.974,00","100,00"})
AADD(aTitulo,{"A","26/03/03","26/03/03","1.974,00","100,00"})

DEFINE MSDIALOG oDlg TITLE cTitulo FROM 122,0 TO 432,600 OF oDlg PIXEL
@ 013,002 TO 154,192 LABEL "Parcelamento" OF oDlg PIXEL
@ 013,195 TO 082,298 LABEL "Títulos" OF oDlg PIXEL
@ 083,195 TO 154,298 LABEL "AIIPM" OF oDlg PIXEL

//scrollbox
@ 019,006 SCROLLBOX oScroll HORIZONTAL VERTICAL SIZE 131,182 OF oDlg
For n:=1 TO Len(aSay)

    bGet := &("{|| '" + aSay[n][1] + "'}")
    cVar := aSay[n][5]
    cGet := "{|u| IIF(PCOUNT()>0, "+ cVar + " := u, "+ cVar + ")}"
    cPict := aSay[n][6]

    TSay():New(nTop,5,bGet,oScroll,,,F.,F.,F.,T.,,);
    GetTextWidth(0,Trim(aSay[n][1]),15,;
    .F.,F.,F.,F.,F.)
    oGet:=TGet():New(nTop-2,40,&cGet,oScroll,,7,cPict,,,,F.,T.,;
    ,.F.,.F.,F.,T.,F.,(cVar),,,T.)
    nTop+=11
Next n

//listbox títulos
@ 019,199 LISTBOX oLbx1 FIELDS HEADER ;
"Parcela","Vencto","Vencto.Real","Valor R$","Qtd.UFESP";
COLSIZES 21,24,33,63,100;
SIZE 095,059 OF oDlg PIXEL
oLbx1:SetArray( aTitulo )
oLbx1:bLine := {||{aTitulo[oLbx1:nAt,1],aTitulo[oLbx1:nAt,2],;
aTitulo[oLbx1:nAt,3],aTitulo[oLbx1:nAt,4],aTitulo[oLbx1:nAt,5]}}

//listbox aiipm
@ 089,199 LISTBOX oLbx2 FIELDS HEADER "AIIPM","Placa","Data Multa" ;

```

```
COLSIZES 24,21,30 SIZE 095,061 OF oDlg PIXEL
oLbx2:SetArray( aAIIPM )
oLbx2:bLine :=
{ || {aAIIPM[oLbx2:nAt,1],aAIIPM[oLbx2:nAt,2],aAIIPM[oLbx2:nAt,3]} }

ACTIVATE MSDIALOG oDlg CENTER ON INIT
EnchoiceBar(oDlg,{ ||oDlg:End() },{ ||oDlg:End() })

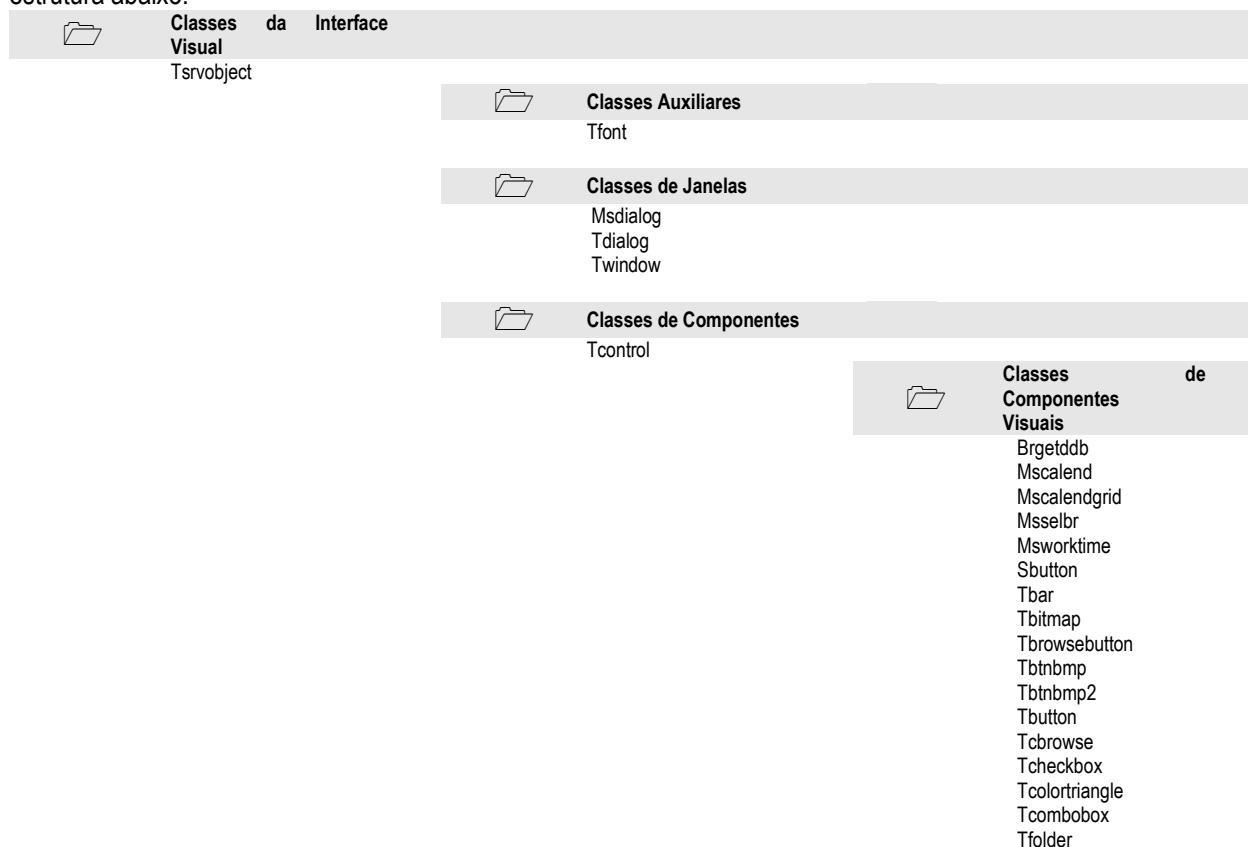
RETURN
```

MÓDULO 06: ADVPL Orientado à objetos

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

16. Componentes da interface visual do ADVPL

A linguagem ADVPL possui diversos componentes visuais e auxiliares, os quais podem ser representados utilizando a estrutura abaixo:



Tfont
Tget
Tgroup
Thbutton
Tibrowser
Tlistbox
Tmenu
Tmenubar
Tmeter
Tmsggraphic
Tmsgbar
Tmultibtn
Tmultiget
Tolecontainer
Tpageview
Tpanel
Tradmenu
Tsbrowse
Tsay
Tscrollbox
Tsimpleeditor
Tslider



Classes de Componentes Visuais

Tsplitter
Ttabs
Ttoolbox
Twbrowse
Vcbrowse

Classes da interface visual

TSRVOBJECT()

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

Classes auxiliares

TFONT()

Descrição	Classe de objetos que define a fonte do texto utilizado nos controles visuais.
------------------	--

Classes de janelas

MSDIALOG()

Descrição	Classe de objetos que deve ser utilizada como padrão de janela para entrada de dados. MSDialog é um tipo de janela diálogo modal, isto é, não permite que outra janela ativa receba dados enquanto esta estiver ativa.
------------------	--

TDIALOG()

Descrição	Classe de objetos do tipo diálogo de entrada de dados, sendo seu uso reservado. Recomenda-se utilizar a classe MSDialog que é herdada desta classe.
------------------	---

TWINDOW()

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

Classes de componentes

TCONTROL()

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

Classes de componentes visuais

BRGETDDB()

Descrição	Classe de objetos visuais do tipo Grid.
------------------	---

MSCALEND()

Descrição	Classe de objetos visuais do tipo Calendário.
------------------	---

MSCALENDGRID()

Descrição	Classe de objetos visuais do tipo Grade de Períodos.
------------------	--

MSSELBR()

Descrição	Classe de objetos visuais do tipo controle - Grid
------------------	---

MSWORKTIME()

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

SBUTTON()

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

TBAR()

Descrição	Classe de objetos visuais do tipo Barra Superior.
------------------	---

TBITMAP()

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

TBROWSEBUTTON()

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

TBTNBMP()

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

TBTNBMP2()

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

TBUTTON()

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

TCBROWSE()

Descrição	Classe de objetos visuais do tipo Grid.
------------------	---

TCHECKBOX()

Descrição	Classe de objetos visuais do tipo controle - CheckBox.
------------------	--

TCOLORTRIANGLE()

Descrição	Classe de objetos visuais do tipo Paleta de Cores.
------------------	--

TCOMBOBOX()

Descrição	Classe de objetos visuais do tipo tComboBox, a qual cria uma entrada de dados com múltipla escolha com item definido em uma lista vertical, acionada por F4 ou pelo botão esquerdo localizado na parte direita do controle. A variável associada ao controle terá o valor de um dos itens selecionados ou no caso de uma lista indexada, o valor de seu índice..
------------------	--

TFOLDER()

Descrição	Classe de objetos visuais do tipo controle - Folder.
------------------	--

TGET()

Descrição	Classe de objetos visuais do tipo controle – tGet, a qual cria um controle que armazena ou altera o conteúdo de uma variável através de digitação. O conteúdo da variável só é modificado quando o controle perde o foco de edição para outro controle.
------------------	---

TGROUP()

Descrição	Classe de objetos visuais do tipo painel – tGroup, a qual cria um painel onde controles visuais podem ser agrupados ou classificados. Neste painel é criada uma borda com título em volta dos controles agrupados.
------------------	--

THBUTTON()

Descrição	Classe de objetos visuais do tipo botão com hiperlink.
------------------	--

TIBROWSER()

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

TLISTBOX()

Descrição	Classe de objetos visuais do tipo controle – tListbox, a qual cria uma janela com itens selecionáveis e barra de rolagem. Ao selecionar um item, uma variável é atualizada com o conteúdo do item selecionado.
------------------	--

TMENU()

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

TMENUBAR()

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

TMETER()

Descrição	Classe de objetos visuais do tipo controle – tMeter, a qual exibe uma régua (gauge) de processamento, descrevendo o andamento de um processo através da exibição de uma barra horizontal.
------------------	---

TMSGRAPHIC()

Descrição	Classe de objetos visuais do tipo controle - Gráfico.
------------------	---

TMSGBAR()

Descrição	Classe de objetos visuais do tipo controle - Rodapé.
------------------	--

TMULTIBTN()

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

TMULTIGET()

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

TOLECONTAINER()

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

TPAGEVIEW()

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

TPANEL()

Descrição	Classe de objetos visuais do tipo controle – tPanel, a qual permite criar um painel estático, onde podem ser criados outros controles com o objetivo de organizar ou agrupar componentes visuais.
------------------	---

TRADMENU()

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

TSBROWSE()

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

TSAY()

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

TSCROLLBOX()

Descrição	Classe de objetos visuais do tipo controle – tScrollbox, a qual permite criar um painel com scroll deslizantes nas laterais (horizontais e verticais) do controle.
------------------	--

TSIMPLEEDITOR()

Descrição	Classe de objetos visuais do tipo controle – tSimpleEditor, a qual permite criar um controle visual para edição de textos com recursos simples, como o NotePad®
------------------	---

TSLIDER()

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

TSPLITTER()

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

TTABS()

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

TTOOLBOX()

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

TWBROWSE()

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

VCBROWSE()

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

Documentação dos componentes da interface visual

Os componentes da interface visual da linguagem ADVPL utilizados neste treinamento estão documentados na seção Guia de Referência, ao final deste material.

Para visualizar a documentação completa de todos os componentes mencionados neste capítulo deve ser acesso o site TDN – TOTVS DEVELOPER NETWORK (tdn.totvs.com.br).

16.1. Particularidades dos componentes visuais

16.1.1. Configurando as cores para os componentes

Os componentes visuais da linguagem ADVPL utilizam o padrão de cores RGB.

As cores deste padrão são definidas pela seguinte fórmula, a qual deve ser avaliada tendo como base a paleta de cores no formato RGB:

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

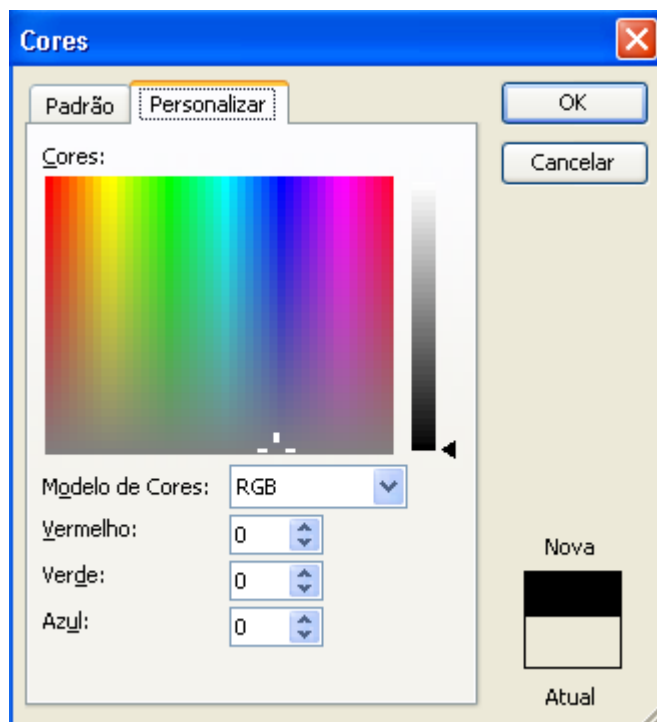


Figura: Paleta de cores no formato RGB

Com base nesta paleta, podemos definir os valores das seguintes cores básicas:

Cor	R	G	B	Valor
Preto	0	0	0	0
Azul	0	0	255	16711680
Verde	0	255	0	65280
Ciano	0	255	255	16776960
Vermelho	255	0	0	255
Rosa	255	0	255	16711935
Amarelo	255	255	0	65535
Branco	255	255	255	16777215

Para atribuir as cores aos objetos visuais devem ser observados os atributos utilizados para estes fins em cada objeto, como por exemplo:

MSDIALOG()

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

TSAY()

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

Função RGB()

A linguagem ADVPL possui a função RGB() a qual retorna o valor da cor a ser definido, de acordo com a parametrização de cada um dos elementos da paleta RGB.

RGB(nRed, nGreen, nBlue)

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

17. Aplicações com a interface visual do ADVPL

A linguagem ADVPL possui interfaces visuais pré-definidas que auxiliam no desenvolvimento de aplicações mais completas, combinando estas interfaces com os componentes visuais demonstrados anteriormente.

Didaticamente as interfaces visuais pré-definidas da linguagem ADVPL podem ser divididas em três grupos:

- **Captura de informações simples ou Multi-Gets;**
- **Captura de múltiplas informações ou Multi-Lines;**
- **Barras de botões**

17.1. Imagens pré-definidas para as barras de botões

Conforme mencionado nos tópicos anteriores, os botões visuais do tipo barra de botões permitem a definição de itens com ações e imagens vinculadas.

Dentre os objetos e funções mencionados, foi citada a `EnchoiceBar()`, a qual permite a adição de botões adicionais através do parâmetro **`aButton`**, sendo que os itens deste array devem possuir o seguinte formato:

Sintaxe: `AADD(aButtons,{cBitMap, bAcao, cTexto})`

Estrutura:

cBitMap	Nome da imagem pré-definida existente na aplicação ERP que será vinculada ao botão.
bAcao	Bloco de código que define a ação que será executada com a utilização do botão.
cTexto	Texto no estilo "tooltip text" que será exibido quando o cursor do mouse for posicionado sobre o botão na barra de ferramentas.

Alguns BitMaps disponíveis:

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

Exemplo:

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

APÊNDICES

BOAS PRÁTICAS DE PROGRAMAÇÃO

18. Arredondamento

Algumas operações numéricas podem causar diferenças de arredondamento. Isso ocorre devido a diferenças no armazenamento de variáveis numéricas nos diversos processadores, diferença esta, inclusive, presente no ADVPL, mesmo antes do surgimento do Protheus.

Para evitar esses problemas de arredondamento, deve ser utilizada a função Round(), principalmente antes de realizar uma comparação e antes de se utilizar a função Int().

Desse modo, assegura-se que o resultado será correto independentemente do processador ou plataforma.

Exemplo 01:

```
If (Valor/30) == 50           // pode ser falso ou inválido
If Round(Valor/30, 0) == 50  // correto
```

Exemplo 02:

```
M->EE8_QTDEM1 := Int(M->EE8_SLDINI/M->EE8_QE)  // pode ser falso ou
inválido
M->EE8_QTDEM1 := Int(Round(M->EE8_SLDINI/M->EE8_QE,10)) // correto
```

19. Utilização de Identação

É obrigatória a utilização da indentação, pois torna o código muito mais legível. Veja os exemplos abaixo:

```
While !SB1->(Eof())
If          mv_par01          ==          SB1->B1_COD
dbSkip()
Loop
Endif
Do Case
Case SB1->B1_LOCAL == "01" .OR. SB1->B1_LOCAL == "02"
TrataLocal(SB1->B1_COD, SB1->B1_LOCAL)
Case SB1->B1_LOCAL == "03"
TrataDefeito(SB1->B1_COD)
Otherwise
TrataCompra(SB1->B1_COD,          SB1->B1_LOCAL)
EndCase
dbSkip()
EndDo
```

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

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

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

    Do Case
        Case SB1->B1_LOCAL == "01" .OR. SB1->B1_LOCAL == "02"
            TrataLocal(SB1->B1_COD, SB1->B1_LOCAL)

        Case SB1->B1_LOCAL == "03"
            TrataDefeito(SB1->B1_COD)

        Otherwise
            TrataCompra(SB1->B1_COD,          SB1->B1_LOCAL)
    EndCase
    dbSkip()

EndDo
```

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

20. 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
```

20.1. Palavras em maiúsculo

A regra é utilizar caracteres em maiúsculo para:

Constantes:

```
#define NUMLINES 60 #define Numpages 1000
```

Variáveis de memória:

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

Campos:

```
SC6->C6_NUMPED
```

Queries:

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

21. Técnicas de programação eficiente

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

- **Funcionalmente correto**
- **Eficiente**
- **Legível**
- **Reutilizável**
- **Extensível**
- **Portável**

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

Criação de funções segundo a necessidade

Observe o código de exemplo:

```
User Function GetAnswer(lDefault)
Local lOk
lOk := GetOk(lDefault)
If lOk
    Return .T.
Else
    Return .F.
Endif
Return nil
```

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

Segundo esse raciocínio, poderia se produzir o seguinte código:

```
User Function GetAnswer(lDefault)
Return If( GetOk(lDefault), .T., .F.)
```

O código acima ainda pode ser aprimorado conforme abaixo:

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

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

Utilização de soluções simples

Simplicidade na criação de instruções torna a programação e até mesmo a execução mais rápida. Considere a linha de código:

```
If nVar > 0 .Or. nVar < 0
```


Se o valor da variável nVar for igual a zero (0) no momento da execução desta linha de código, ambas as comparações separadas pelo operador lógico .Or. serão efetuadas: Após ser avaliada, a primeira comparação irá falhar. A segunda comparação será então avaliada e falhará também. Como resultado, o código existente dentro da estrutura de fluxo If não será executado. Tal código somente será executado quando o valor desta variável for maior OU menor do que zero. Ou seja, sempre que for DIFERENTE de zero, o que torna a linha a seguir mais eficiente:

```
If nVar != 0
```

Este tipo de alteração torna o código mais legível e o processamento mais rápido, evitando a avaliação de instruções desnecessariamente.

Existem outras situações onde a simplificação pode ser utilizada. A expressão de avaliação a seguir:

```
If cVar == "A" .Or. cVar == "B" .Or cVar == "C" .Or. cVar == "D"
```

Pode ser substituído pelo operador de contenção:

```
If cVar $ "ABCD"
```

Opção por flexibilidade

A melhor solução é aquela que envolve o problema imediato e previne problemas no futuro. Considere o exemplo:

```
@nRow,nCol PSAY cVar Picture "!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
```

Exceto contando-se os caracteres, não existe maneira de saber se o número de caracteres de exclamação é o esperado. Enquanto isto é um problema, existem algo mais grave. A expressão de picture é estática. Se no futuro for necessário ajustar o tamanho da variável cVar, será necessário localizar todos os lugares no código onde esta máscara de picture está sendo utilizada para ajuste manual.

Existe uma opção de solução de auto-ajuste disponível que é fácil de digitar e tem a garantia de executar a tarefa igualmente (tornar todos os caracteres maiúsculos):

```
@nRow,nCol PSAY cVar Picture "@!"
```

Opção da praticidade ao drama

Se a solução parece complexa, provavelmente é porque o caminho escolhido está levando a isso. Deve-se sempre se perguntar porque alguém desenvolveria uma linguagem que requisite tantos comandos complicados para fazer algo simples. Na grande maioria dos casos, existe uma solução mais simples. O exemplo abaixo deixa isso bem claro:

```
@ 10,25 Say Substr(cCep,1,5) + "-" + Substr(cCep,6,3) Picture
"!!!!!!!!!!"
```

Este código pode ser escrito de uma forma muito mais simples, conforme demonstrado abaixo:

```
@ 10,25 Say cCep Picture "@R 99999-999"
```

Utilização de operadores de incremento/decremento

Utilizados devidamente, os operadores de incremento e decremento tornam o código mais fácil de ler e possivelmente um pouco mais rápidos. Ao contrário de escrever adições simples como:

```
nVar := nVar + 1
nVar := nVar -1
```

Pode-se escrevê-las assim:

```
++nVar
--nVar
```

Deve-se apenas tomar cuidado com a precedência destes operadores, pois o "++" ou o "--" podem aparecer antes ou depois de uma variável, e em alguns casos quando a variável for utilizada dentro de uma expressão, a prefixação ou sufixação destes operadores afetará o resultado. Para maiores detalhes, consulte a documentação de operadores da linguagem ADVPL.

Evitar passos desnecessários

Existe uma diferença entre um bom hábito e perda de tempo. Algumas vezes estes conceitos podem estar muito próximos, mas um modo de diferenciá-los é balancear os benefícios de realizar alguma ação contra o problema que resultaria se não fosse executada. Observe o exemplo:

```
Local nCnt := 0
For nCnt := 1 To 10
<código>
Next nCnt
```

Inicializar a variável no momento da declaração não é um problema. Se o 0 fosse necessário no exemplo, teria sido útil a inicialização na declaração. Mas neste caso a estrutura de repetição **For...Next** atribui o seu valor imediatamente com 1, portanto não houve ganho em atribuir a variável com 0 no começo.

Neste exemplo não há nenhum ponto negativo e nada errado ocorrerá se a variável não for inicializada, portanto é aconselhável evitar este tipo de inicialização, pois não torna o código mais seguro e também não expressa a intenção do código mais claramente.

Porém note este exemplo, onde a variável não é inicializada:

```
Local nCnt
While ( nCnt++ < 10 )
<código>
EndDo
```

Em ADVPL, variáveis não inicializadas sempre tem seu valor contendo nulo (nil) a princípio, o que fará com que uma exceção em tempo de execução aconteça quando a instrução de repetição while for executada.

Diferentemente do primeiro exemplo, onde a inicialização da variável não fazia diferença alguma, neste segundo exemplo a inicialização é absolutamente necessária. Deve-se procurar inicializar variáveis numéricas com zero (0) e variáveis caracter com string nula (""), apenas quando realmente necessário.

Utilização de alternativas

Quando se está trabalhando em uma simples rotina, deve-se tomar algum tempo para explorar duas ou três diferentes abordagens. Quando se está trabalhando em algo mais complexo, deve-se planejar prototipar algumas a mais. Considere o seguinte código:

```
If cHair = "A"
  Replace hair With "Loira"
Else
  If cHair = "B"
    Replace hair With "Morena"
  Else
    If cHair = "C"
      Replace hair With "Ruiva"
    Else
      If cHair = "D"
        Replace hair With "Grisalho"
```

```
        Else
            Replace hair With "Preto"
        Endif
    Endif
Endif
```

Um código de uma única letra, (A até E), foi informado para indicar a cor de cabelo. Este código foi então convertido e armazenado como uma string. Pode-se notar que a cor "Preto" será atribuída se nenhuma outra opção for verdadeira.

Uma alternativa que reduz o nível de indentação torna o código mais fácil de ler enquanto reduz o número de comandos replace:

```
Do Case
    Case cHair == "A"
        cColor := "Loira"
    Case cHair == "B"
        cColor := "Morena"
    Case cHair == "C"
        cColor := "Ruiva"
    Case cHair == "D"
        cColor := "Grisalho"
    OtherWise
        cColor := "Preto"
EndCase

Replace hair With cColor
```

Utilização de arquivos de cabeçalho quando necessário

Se um arquivo de código criado se referencia a comandos para interpretação e tratamento de arquivos XML, este deve se incluir o arquivo de cabeçalho próprio para tais comandos (XMLXFUN.CH no exemplo). Porém não deve-se incluir arquivos de cabeçalho apenas por segurança. Se não se está referenciando nenhuma das constantes ou utilizando nenhum dos comandos contidos em um destes arquivos, a inclusão apenas tornará a compilação mais demorada.

Constantes em maiúsculo

Isto é uma convenção que faz sentido. Em ADVPL, como em C por exemplo, a regra é utilizar todos os caracteres de uma constante em maiúsculo, a fim de que possam ser claramente reconhecidos como constantes no código, e que não seja necessários lembrar onde foram declarados.

Utilização de indentação

Este é um hábito que todo programador deve desenvolver. Não consome muito esforço para manter o código alinhado durante o trabalho, porém quando necessário pode-se utilizar a ferramenta TOTVS DevStudio para a re-indentação de código. Para maiores detalhes, consulte a documentação sobre a indentação de códigos fontes disponível nos demais tópicos deste material.

Utilização de espaços em branco

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

Quebra de linhas muito longas

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

Capitulação de palavras-chave

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

Avaliando o código a seguir:

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

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

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

Utilização da Notação Húngara

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

Utilização de nomes significantes para variáveis

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

Criar variáveis como `nNumero` ou **dData** também não ajudam na identificação. A Notação Húngara já está sendo utilizada para isso e o objetivo do nome da variável deveria ser identificar sua utilização, não o tipo de dado utilizado. Deve-se procurar substituir tais variáveis por algo como **nTotal** ou **dCompra**.

O mesmo é válido para nomes de funções, que devem descrever um pouco sobre o que a função faz. Novamente nomes extremamente curtos não são aconselháveis.

Utilização de comentários

Comentários são muito úteis na documentação de programas criados e para facilitar a identificação de processos importantes no futuro e devem sempre ser utilizados.

Sempre que possível, funções criadas devem ter uma breve descrição do seu objetivo, parâmetros e retorno. Além de servir como documentação, os comentários embelezam o código ao separar as funções umas das outras. Os comentários devem ser utilizados com bom senso, pois reescrever a sintaxe ADVPL em português torna-se apenas perda de tempo:

```
If nLastKey == 27 // Se o nLastKey for igual a 27
```

Criação de mensagens sistêmicas significantes e consistentes

Seja oferecendo assistência, exibindo mensagens de aviso ou mantendo o usuário informado do estado de algum processo, as mensagens devem refletir o tom geral e a importância da aplicação. Em termos gerais, deve-se evitar ser muito informal e ao mesmo tempo muito técnico.

```
"Aguarde.      Reindexando      (FILIAL+COD+      LOCAL)      do      arquivo:  
\DADOSADV\SB1990.DBF"
```

Esse tipo de mensagem pode dar informações demais para o usuário e deixá-lo sentindo-se desconfortável se não souber o que significa "reindexando", etc. E de fato, o usuário não devia ser incomodado com tais detalhes. Apenas a frase "Aguarde, indexando." funcionaria corretamente, assim como palavras "processando" ou "reorganizando".

Outra boa idéia é evitar a referencia a um item corrente de uma tabela como um "registro":

```
"Deletar este registro?"
```

Se a operação estiver sendo efetuada em um arquivo de clientes, o usuário deve ser questionado sobre a remoção do cliente corrente, se possível informando valores de identificação como o código ou o nome.

Evitar abreviação de comandos em 4 letras

Apesar do ADVPL suportar a abreviação de comandos em quatro letras (por exemplo, repl no lugar de replace) não há necessidade de utilizar tal funcionalidade. Isto apenas torna o código mais difícil de ler e não torna a compilação mais rápida ou simples.

Evitar "disfarces" no código

Não deve-se criar constantes para expressões complexas. Isto tornará o código muito difícil de compreender e poderá causar erros primários, pois pode-se imaginar que uma atribuição é efetuada a uma variável quando na verdade há toda uma expressão disfarçada:

```
#define NUMLINES aPrintDefs[1]
#define Numpages aPrintDefs[2]
#define ISDISK   aReturn[5]

If ISDISK == 1
    NUMLINES := 55
Endif

Numpages += 1
```

A impressão que se tem após uma leitura deste código é de que valores estão sendo atribuídos às variáveis ou que constantes estão sendo utilizadas. Se o objetivo é flexibilidade, o código anterior deve ser substituído por:

```
#define NUMLINES 1
#define Numpages 2
#define ISDISK   5

If aReturn[ISDISK] == 1
    aPrintDefs[ NUMLines ] := 55
Endif

aPrintDefs[ Numpages ] += 1
```

Evitar código de segurança desnecessário

Dada sua natureza binária, tudo pode ou não acontecer dentro de um computador. Adicionar pedaços de código apenas para "garantir a segurança" é freqüentemente utilizado como uma desculpa para evitar corrigir o problema real. Isto pode incluir a checagem para validar intervalos de datas ou para tipos de dados corretos, o que é comumente utilizando em funções:

```
Static Function MultMalor( nVal )  
If ValType( nVal ) != "N"  
    nVal := 0  
Endif  
Return ( nVal * nVal )
```

O ganho é irrisório na checagem do tipo de dado do parâmetro já que nenhum programa corretamente escrito em execução poderia enviar uma string ou uma data para a função. De fato, este tipo de "captura" é o que torna a depuração difícil, já que o retorno será sempre um valor válido (mesmo que o parâmetro recebido seja de tipo de dado incorreto). Se esta captura não tiver sido efetuada quando um possível erro de tipo de dado inválido ocorrer, o código pode ser corrigido para que este erro não mais aconteça.

Isolamento de strings de texto

No caso de mensagens e strings de texto, a centralização é um bom negócio. Pode-se colocar mensagens, caminhos para arquivos, e mesmo outros valores em um local específico. Isto os torna acessíveis de qualquer lugar no programa e fáceis de gerenciar.

Por exemplo, se existe uma mensagem comum como **"Imprimindo, por favor aguarde..."** em muitas partes do código, corre-se o risco de não seguir um padrão para uma das mensagens em algum lugar do código. E mantê-las em um único lugar, como um arquivo de cabeçalho, torna fácil a produção de documentação e a internacionalização em outros idiomas.