Protheus 10

Educação Corporativa

Programação em ADVPL

microsiga

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

Sumário

LINGUAGEM DE PROGRAMACÃO ADVPL

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

Para rodar um programa desenvolvido em AdvPl é preciso antes de mais nada escrevê-lo e compilá-lo. Isto é feito no IDE (Integrated Development Environment, ou Ambiente Integrado de Desenvolvimento). O objetivo do IDE é facilitar a tarefa de escrever programas: através de cores indica se a palavra escrita é uma instrução, uma variável ou um comentário; organiza a biblioteca de programas em projetos; gera modelos de programas de cadastramento, relatórios, consultas, etc; faz o debug, ou seja, permite que se teste o programa passo a passo, verificando o conteúdo das variáveis, marcando pontos de parada, etc.

Após compilar o programa, o resultado é um objeto. Este objeto é então interpretado, ou seja, é carregado na memória e o Protheus passa a executá-lo. Ou seja, para rodar um programa é necessário que o Protheus esteja presente na memória. O Protheus, por sua vez, é um sistema multi-camada. É dividido em Remote, Server e Dados.

O Remote processa a parte da estação, basicamente tela e teclado. Pode estar gravado no Servidor e ser carregado via rede para a memória da estação. Ou, de preferência, deve ficar armazenado no HD da estação. Pode também ser carregado pelo Internet Explorer, rodando dentro do Browser.

O Server é o responsável pelo processamento dos objetos chamados pelo usuário. É ele que se comunica com o Banco de Dados, seja diretamente se for DBFs ou Ctree, ou via Top Connect em caso de SQL. Uma vez terminado o processamento do objeto chamado, o mesmo é descartado da memória. Ou seja, o Protheus é um sistema que pode crescer de forma ilimitada pois os objetos, armazenados em um repositório de objetos, praticamente não ocupam espaço no HD.

A linguagem propriamente dita

Como é sabido, uma linguagem de programação é um conjunto de comandos e funções que definem ao computador o passo a passo de uma tarefa.

Basicamente os comandos se dividem em três tipos:

- aritméticos: + - / * somar subtrair dividir e multiplicar.
- entrada e saída: leitura e gravação em uma mídia eletrônica (disco, memory key, etc), da tela e impressão (papel ou tela) e também o comando de atribuição, que move os dados na memória de um campo para outro, através do :=
- comandos lógicos, que são na verdade, aqueles que dão "inteligência" ao computador. Liderados pelo IF, podem-se apresentar de outras formas: While, For e Do Case.

Os operadores usados no AdvPl são os seguintes:

Matemáticos

** ou ^ Exponenciação.
* Multiplicação.
/ Divisão.
% Módulo (resto da divisão).
+ Adição ou sinal positivo.
- Subtração ou sinal negativo.

Relacionais

< Menor que.
> Maior que.
== Igual a.
<= Menor que ou igual a.
>= Maior que ou igual a.
<> ou # ou != Diferente.

Lógicos

.Not. ou ! "Não" lógico.
.And. "E" lógico.
.Or. "Ou" lógico.

Atribuição

:= Permite atribuir um valor a uma variável.
+= Adiciona antes de atribuir.
-= Subtrai antes de atribuir.
*= Multiplica antes de atribuir.
/= Divide antes de atribuir.
^= ou **= Eleva antes de atribuir.
%= Calcula o módulo antes de atribuir.

Incremento/Decremento

++x Soma um ao valor de x e então retorna o valor de x já atualizado.
X++ Retorna o valor de x e então soma um ao valor de x.
--x Subtrai um do valor de x e então retorna o valor de x já atualizado.
x-- Retorna o valor de x e então subtrai um do valor de x.

Strings

x+y Concatenação dos strings x e y.
x-y Concatenação dos strings, mas os espaços em branco à direita do primeiro operando (x) serão transferidos para o fim do string resultante da concatenação.
x $ y Retorna verdadeiro se o conteúdo da variável x estiver contido no conteúdo da variável y.

Especiais

&cVariavel	Operador macro.
||	Indica uma lista de argumento em um bloco de código.
()	Função ou agrupamento em fórmula numérica ou macro operador.
[]	Referência a um elemento de uma matriz.
{}	Delimitador do bloco de código ou atribuição de valores literais para uma matriz.
->	Referência a um alias, sendo um indicador.
@	Passagem de parâmetros por referência.
;	Separa instruções numa mesma linha.
:	Usando como operador "send" quando acessa nova classe de objeto, atribuição a objeto.

O que você aprendeu neste capítulo

Neste capítulo nós ficamos sabendo sobre alguns conceitos básicos do Advpl e seus operadores.

Próximo Passo

No próximo capítulo, iremos aprender: variáveis, arrays e funções

Anotações

VARIÁVEIS

O que você irá aprender neste capítulo

Neste capítulo, veremos 3 conceitos importantes: variáveis, arrays e funções.

Rotinas Abordadas

Variáveis.
Arrays(Matriz).
Funções.

Variáveis

São campos na memória manipulados pelo programa. No AdvPl estes campos não são rigorosamente tipados, ou seja, mesmo que um campo seja numérico ele pode receber um texto.

De qualquer forma ao se definir as variáveis, elas podem assumir os seguintes tipos:

Numéricas: para cálculos, podendo ter até 14 inteiros e 8 decimais.

Caractere: para textos.

Lógicas: podem assumir a condição de Verdadeiro (.T.) ou Falso (.F.). São usadas como chaves de decisão.

Data: devido ao tratamento especifico dado aos dias/meses/anos há um tipo próprio que permite somar e subtrair um certo número de dias à uma data ou mesmo fazer comparações entre elas.

A notação húngara sugere que iniciemos o nome do campo com uma letra minúscula indicando seu tipo.

Exemplo:

cTexto, nValor, dVencimento, lAchou.

Para atribuir um valor à variável usa-se :=

Exemplo:

cTexto := "casa" as aspas indicam que se trata de um texto

nValor := 123,45

dVencimento := CtoD("25/12/05") CtoD converte o texto em data

lAchou := .T. lAchou fica verdadeiro.

O nome da variável deve ser absolutamente significativo.

Exemplo: um campo para armazenar o custo médio deve ter uma nomenclatura similar a esta: nCustoMedio

Dica

Embora o Protheus possua a limitação de dez posições para o nome da variável, esta pode ser desprezada nos programas, desde que não gerem duplicidade de símbolos.

Exemplo: nCustoMedio1 e nCustoMedio2 ambas serão truncadas em dez posições e serão entendidas como nCustoMedio.

O AdvPl não é uma linguagem de tipos rígidos para variáveis, ou seja, não é necessário informar o tipo de dados que determinada variável irá conter no momento de sua declaração, e o seu valor pode mudar durante a execução do programa. Também não há necessidade de declarar variáveis em uma seção específica do seu código fonte, embora seja aconselhável declarar todas as variáveis necessárias no começo, tornando a manutenção mais fácil e evitando a declaração de variáveis desnecessárias.

Para declarar uma variável, deve-se utilizar um identificador de escopo, seguido de uma lista de variáveis separadas por vírgula (,).

Um identificador de escopo é uma palavra-chave que indica a que contexto do programa a variável declarada pertence. O contexto de variáveis pode ser local (visualizadas apenas dentro do programa atual "função"), público (visualizadas por qualquer outro programa), entre outros.

Mais detalhes:

Anotações

Escopo das Variáveis

Um programa, que sempre tem como sufixo de seu nome a sigla .prw, é na verdade um conjunto de funções e de dentro dele são chamadas outras, que estão no Repositório. Para organizar este ambiente e, de um lado, impedir que uma função destrua as variáveis de outra e, por outro, permitir que haja uma integração entre elas, cada variável tem o seu escopo.

Variáveis Locais: são definidas dentro da função e somente são visualizadas e podem ser atualizadas por comandos internos à função.

Variáveis Privadas: podem ser visualizadas e alteradas pela função que a definiu e por todas as outras que ela chamar. Neste caso não há necessidade de passá-la como parâmetro.

Variáveis Públicas: Podem ser visualizadas e atualizadas em qualquer função. São normalmente variáveis definidas na chamada do sistema para uso geral. Ex: dDataBase, é a data digitada pelo usuário na chamada do sistema. O mesmo com cFilial.

Variáveis Estáticas: Podem ser visualizadas e atualizadas dentro do PRW. Seu uso é raro.

Exercício 01 – Digite no Ide o código abaixo, salve o arquivo com o nome de Exerc01, compile e execute a função Exerc01.

Observação 1: Dentro de um arquivo, é possível ter várias funções. Portanto, depois do arquivo compilado, quem deverá ser executado é a função, que pode ter o mesmo nome do arquivo. Prw. Porem, não se esqueça que se executa a função e não o arquivo.

Observação 2: No Advpl, para comentar uma linha usa-se //
Para comentar várias linhas seguidas usa-se:

```
/*
User Function Exerc01
nResultado := 250 * (1 + (nPercentual / 100))
Alert(nResultado)
Return( )

*/

//Considere as linhas de código de exemplo:
#Include "Rwmake.ch"
User Function Exerc01
nResultado := 250 * (1 + (nPercentual / 100))
Alert(nResultado)
Return( )
```

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

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

Neste exemplo, as variáveis são declaradas previamente utilizando o identificador de escopo local. Quando a linha de cálculo for executada, o erro de variável não existente não ocorrerá.

Porém, variáveis não inicializadas têm sempre o valor default nulo (Nil) e este valor não pode ser utilizado em um cálculo, pois também gerará erros de execução (nulo não pode ser dividido por 100).

A resolução deste problema é efetuada inicializando-se a variável por meio de uma das formas:

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

A diferença entre o último exemplo e os dois anteriores é que a variável é inicializada no momento da declaração.

Nos dois primeiros exemplos, a variável é primeiro declarada e então inicializada em uma outra linha de código.
Deve-se utilizar o operador de atribuição (:= ou somente =).

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

Uma vez que um valor lhe seja atribuído, o tipo de dado de uma variável é igual ao tipo de dado do valor atribuído.

Ou seja, uma variável passa a ser numérica se um número lhe é atribuído, passa a ser caracter se uma string de texto lhe for atribuída, etc.

Porém, mesmo que uma variável seja de determinado tipo de dado, pode-se mudar o tipo da variável atribuindo outro tipo a ela:

Exercício 02 – Exercitando a declaração de variáveis

Observação: Debug o programa juntamente com o Watchs.

```
#Include "Rwmake.ch"
User Function Exerc02

Local xVariavel // Declara a variável inicialmente com valor nulo.

 xVariavel := "Agora a variável é caracter..."
 Alert("Valor do Texto:" + xVariavel)
 xVariavel := 22 // Agora a variável é numérica.
 Alert(cValToChar(xVariavel))

 xVariavel := .T. // Agora a variável é lógica.
 If xVariavel
   Alert("A variável tem valor verdadeiro...")
 Else
Alert("A variável tem valor falso...")
 Endif
Set Date British

 xVariavel := Date() // Agora a variável é data.
 Alert("Hoje é:" + DtoC(xVariavel))

 xVariavel := nil // Nulo novamente.
 Alert("Valor nulo:" + xVariavel)

 Return( )
```

Este programa troca os valores da variável e exibe seu conteúdo para o usuário por meio da função alert. Essa função recebe um parâmetro que deve ser do tipo string de caracter, por isso dependendo do tipo de dado da variável xVariavel é necessário fazer uma conversão antes.

Apesar dessa flexibilidade de utilização de variáveis, deve-se tomar cuidado na passagem de parâmetros para funções ou comandos e na concatenação (ou soma) de valores.

A tentativa de soma de tipos de dados diferentes gera erro de execução do programa. "type mismatch on +".

Executando o caso do valor nulo, para os demais se deve sempre utilizar funções de conversão quando se necessita concatenar tipos de dados diferentes.

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

If xVariavel
é o mesmo que
If xVariavel = .T.

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

Observação: Durante o curso será abordado funções com detalhes.

Anotações

MATRIZES - ARRAYS

ARRAYS

O array, matriz ou vetor também é um tipo de variável. Nele pode-se armazenar uma seqüência de dados, identificados por um nome do array e o número do elemento. Cada elemento, por sua vez, pode ser um novo array. É como se tivéssemos um arquivo ou uma planilha Excel na memória, com um processamento extremamente rápido. Considerando o tamanho das memórias hoje em dia pode-se criar arrays bastante grandes.

O array é definido com tamanho fixo ou variável:

Exemplo:

aTabela := Array(30) aTabela tem 30 elementos.
aTabela := {} array de tamanho indefinido ou seja pode receber tantos elementos quantos couberem na memória.
aTabela := {1,2,4) array com 3 elementos e seus respectivos valores.
aTabela := {1,2,"casa") os elementos podem ser de tipos diferentes.
aTabela := Array(3,2) o array terá 3 linhas, cada uma com 2 colunas.
aTabela := Array(3,2,2) é um array multidimensional, cada linha tem 2 colunas, que por sua vez tem mais 2 elementos cada:

aTabela:= {{{L1,C1,X1},{L1,C1,X2}},{{L1,C2,XA},{L1,C2,XB}},;

{{L2,C1,XC},{L2,C1,XD}},{{L2,C2,XE},{L2,C2,XF}}.....

Para evocar um elemento usa-se colchetes:

aTabela[1] é o primeiro elemento de um array simples.
aTabela[i] é um elemento identificado pela variável i.

Exercício 03 – Exercitando o uso de Matriz

```
Local aLetras     // Declaração da variável.
aLetras:={"A","B","C"} // Atribuição da matriz à variável.
Alert(aLetras[2])                    //Exibe o segundo elemento da matriz.
Alert(cValtoChar(Len(aLetras)))   //Exibe o tamanho da matriz.
```

Observação 1: Lembre-se de utilizar o IDE, para compilar e debugar o seu programa.

Observação 2: Enquanto em outras linguagens como C ou Pascal é necessário alocar memória para cada elemento de uma matriz (o que tornaria a utilização de "ponteiros" necessária), o Advpl encarrega-se de gerenciar a memória e torna simples a adição de elementos a uma matriz, utilizando a função aAdd:

```
aAdd(aLetras,"D")        //Adiciona o quarto elemento ao final da matriz.
Alert(aLetras[4])        //Exibe o quarto elemento.
Alert(aLetras[5])        //Erro! Não há um quinto elemento na matriz.
```

Matrizes como Estruturas

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

aFunct1 := {"Pedro",32,.T.}

Esta matriz contem uma string, um número e um valor lógico.

Em outras linguagens como C ou Pascal, este "pacote" de informações pode ser chamado como um "struct" (estrutura em C, por exemplo) ou um "record" (registro em Pascal, por exemplo).

Como se fosse na verdade um registro de um banco de dados, um pacote de informações construído com diversos campos, cada campo tendo um pedaço diferente de dado.

Suponha-se que, no exemplo anterior, o array aFunct1 contenha informações sobre o nome de uma pessoa, sua idade e sua situação matrimonial. Os seguintes #defines podem ser criados para indicar cada posição dos valores dentro da matriz:

```
#define FUNCT_NOME   1
#define FUNCT_IDADE  2
#define FUNCT_CASADO 3
```

E considere mais algumas matrizes para representar mais pessoas:

```
aFunct2 := {"Maria"  , 22, .T.}
aFunct3 := {"Antônio", 42, .F.}
```

Os nomes podem ser impressos assim:

```
Alert(aFunct1[FUNCT_NOME])
Alert(aFunct2[FUNCT_NOME])
Alert(aFunct3[FUNCT_NOME])
```

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

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

Que é equivalente a isso:

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

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

```
For nCount := 1 To Len(aFuncts)
Alert(aFuncts[nCount,FUNCT_NOME])
// O acesso a elementos de uma matriz multidimensional
```

// pode ser realizado também desta forma:

// aFuncts[nCount][FUNCT_NOME]

Next nCount

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

FUNÇÕES: a maior parte das rotinas que queremos escrever em programas é composta de um conjunto de comandos, rotinas estas que se repetem ao longo de todo o desenvolvimento. Uma Função nada mais é do que um conjunto de comandos. Para ser usada basta chamá-la pelo seu nome. Para tornar uma Função mais flexível ao chamá-la pode-se passar parâmetros. Estes parâmetros contem dados e informações que influem no processamento da função. Os parâmetros no AdvPl são posicionais, ou seja, na sua passagem não importa o nome da variável e sim a sua posição. Assim podemos chamar uma função escrevendo:

Calcula(para,parb,parc)

E a função estar escrita

Calcula(x,y,z)

Neste caso, x assume o valor de para, y de parb e z de parc.

A função também tem a faculdade de retornar uma variavel, podendo inclusive ser um array. Para tal encerra-se a Função com

Return(campo)

Assim A := Calcula(para,parb,parc) atribui à A o conteúdo do retorno da função Calcula.

No Advpl existem milhares de Funções escritas pelo pessoal de Tecnologia, pelos analistas de suporte e pelos próprios usuários. Existe um ditado que diz que "vale mais um programador que conhece todas as funções disponíveis em uma linguagem do que aquele que reinventa a roda a cada novo programa". Mas conhecer todas as funções vem com o tempo. No DEM mais de 500 estão documentadas. Aqui no curso não abordaremos mais do que 100. Cabe a cada um estudá-las quando delas necessitar.

No AdvPl até os programas chamados do menu são funções. Num repositório não pode haver funções com o mesmo nome. Para contornar este problema as funções escritas pelo usuário têm o prefixo U_ de User Function.

O que você aprendeu neste capítulo

Neste capítulo aprendemos sobre variáveis, arrays(matriz) e falamos sobre funções.

Próximo Passo

No próximo capítulo, iremos aprender as estruturas de controle:

ESTRUTURA DE CONTROLE

O que você irá aprender neste capítulo

Neste capítulo, nós iremos aprender as estruturas de controle da linguagem.

Rotinas Abordadas

For ... Next
While...End
If….Elseif….Else……Endif
Do Case...case...Otherwise...End Case

For Next

Permite definir um laço de comandos

For nVar := nExpr1 To Expr2 [Step nExpr3]
<comandos>…
[Exit]
<comandos>…
[Loop]
Next
nVar variável de comando do laço
nExpr1 valor inicial da variável de controle
nExpr2 valor final da variável de controle
nExpr3 incremento (n) ou decremanto (-n) da variável de controle (default 1)

Exercício 04 – Entendendo a estrutura For – Next.

```
User Function TstFor()
Local i
For i := 1 To 10
   MsgAlert(i)
Next

Return

//————————————————————————————————————————————————————//
User Function TstFor1()

Local i
Local nIni, nFim

nIni := 100
nFim := 120

For i := nIni To nFim Step 2
   MsgAlert(i)
```

```
   If i > 110
      Exit     // Break também encerra.
   EndIf
Next

Return

//————————————————————————————————————————————————————————//
User Function TstFor2()

Local i
Local nIni, nFim

nIni := 1
nFim := 10

For i := nFim To nIni Step -1
   MsgAlert(i)
Next

Return

//————————————————————————————————————————————————————————//
User Function TstFor3()

Local i
Local j

For i := 20 To 25
   MsgAlert("i=" + Str(i))
   For j := 1 To 5
      MsgAlert("i=" + Str(i) + "   j=" + Str(j))
   Next
Next

Return
```

Observação: Lembre-se de utilizar o IDE, para compilar e debugar o seu programa.

——-

Anotações

While...End

O While é um comando de programação estruturada que permite que as instruções contidas entre While e o associado end sejam repetidas enquanto uma determinada condição permanecer verdadeira.

```
While <expressão lógica>
        <comandos....>
[Exit]
        <comandos...>
[Loop]
End
```

Exercício 05 – Entendendo a estrutura While – End.

```
User Function TstWhile()

Local i := 1

While i <= 10

  MsgAlert(i)
  i++

End

Return
```

———-

Anotações

If....Elseif....Else.....Endif

O comando if é um commando de programação que permite executar condicionalmente um bloco de instruções, também conhecido como desvio condicional. Cada if deve obrigatoriamente terminar com um endif.

```
If<Condição>
<Comandos>
Elseif<Condição>
<Comandos>
Else
<Comandos>
Endif
```

Exercício 06 – Entendendo a estrutura If – Elseif e Endif.

```
User Function TstIf()

Local nX := 10

If nX > 5
  MsgAlert("Maior")
EndIf

Return

User Function TstElse()

Local nX := 10
Local cMsg

If nX < 5
  cMsg := "Menor"
 Else
  cMsg := "Maior"
EndIf

MsgAlert(cMsg)

Return

————————————————————————————————————————————————————————————-
// Avalia a partir do IF e depois os ElseIf. Ao encontrar a primeira
// condiçao verdadeira, executa vai para o EndIf.
————————————————————————————————————————————————————————————-

User Function TstElseIf()

Local cRegiao := "NE"
Local nICMS

If cRegiao == "SE"
```

```
  nICMS := 18
 ElseIf cRegiao == "NE"
  nICMS := 7
 Else
  nICMS := 12
EndIf

MsgAlert(nICMS)

Return
```

Anotações

Do Case...case...Otherwise...End Case

O comando Do Case..EndCase é estruturado e seleciona apenas um caminho entre um conjunto de caminhos alternativos para o curso do fluxo de execução.

Exercício 07 – Entendendo a estrutura Case – EndCase.

```
User Function TstCase()

Local nOpc := 2

Do Case
  Case nOpc == 1
     MsgAlert("Opção 1 selecionada")
  Case nOpc == 2
     MsgAlert("Opção 2 selecionada")
  Case nOpc == 3
     MsgAlert("Opção 3 selecionada")
  Otherwise
     // Otherwise é opcional.
     MsgAlert("Nenhuma opção selecionada")
EndCase

Return
```

O que você aprendeu neste capítulo

Neste capítulo aprendemos sobre as estruturas de controle da linguagem.

Próximo Passo

No próximo capítulo, iremos aprender: Macro substituição, lista de expressões e bloco de código.

Anotações

MACRO SUBSTITUIÇÃO

O que você irá aprender neste capítulo

Neste capítulo, nós iremos aprender sobre: Macro substituição, lista de expressões e bloco de código.

Rotinas Abordadas

Macro
Lista
Bloco de Código

Macro

Macro-substituição

A Macro-substituição trata o conteúdo do campo como se ele fosse uma linha de código de um programa. Ocorre que este conteúdo pode ser lido de um arquivo externo (SX3, SX7, SI5, SM4, etc), arquivo este que pode ser alterado pelo usuário. Estas expressões são encontradas no dicionário de dados, nos campos de validação e inicialização, nos gatilhos, no arquivo de lançamento padronizado, na folha de pagamento, no arquivo de fórmulas, entre outros. Quando se usa a Macro-substituição em um programa deve ela ser sempre precedida pela leitura da expressão em um arquivo.

Exemplo

```
User Function TstMacro()

Local cCampo
Local cCalc

dbSelectArea("SA1")
cCampo := "A1_Nome"

MsgAlert(cCampo + ":" + &cCampo)     // Mostra o conteudo do campo A1_Nome.

dbSelectArea("SA2")
cCampo := "A2_Nome"

MsgAlert(cCampo + ":" + &cCampo)     // Mostra o conteudo do campo A2_Nome.

cCalc := "1 + 1"

MsgAlert(cCalc + ":" + Str(&cCalc))   // Mostra o resultado do caculo 1+1.

Return Nil
```

Outro Exemplo

```
User Function Tesmacro( )
A1_MOEDA1:= "MOEDA1"
A1_MOEDA2:= "MOEDA2"
A1_MOEDA3:= "MOEDA3"

cCAMPO:= "A1_MOEDA"
cESCOLHA:= "3"
cMacro:=&(cCampo+cEscolha)
MsgAlert(cMacro)
```

Lista de expressões

O AdvPL aceita que se escreva vários comandos em uma única lista de expressões, abrindo maiores possibilidades no uso de validações e gatilhos onde só temos o espaço de uma linha para escreve-la. Nestes casos quando se tem um código mais extenso, a solução é escrever-se uma função, que pode ter um tamanho ilimitado, mas por outro lado precisa ser compilada.

Bloco de código

O Bloco de Código permite que se execute uma expressão externa em tempo de execução exatamente da mesma forma que a Macro-substituição, mas com uma vantagem: permite que se passe parâmetros, como se fosse uma função. Para executar um Bloco de Código usa-se a função Eval, que retorna o resultado do processamento do Bloco.

Outra vantagem do uso do Bloco de Código está na existência das funções aEVAL e DBEval. A aEval executa o Bloco para todos os elementos do array sem necessidade de nenhum comando adicional.
A DBEval processa todos os registros de um arquivo.

Dica

A função Eval( ) retorna o valor da última expressão dentro do bloco. Um bloco de código pode retornar um valor de qualquer tipo.

Exemplo de um Bloco:

```
User Function Bcod()
x:=20
y:=24
bBloco:={|x,y|If(x>y,"o valor de x é maior,"o valor de y é maior")}
MsgAlert(Eval(bBloco, x,y))

Return( )
```

——-

```
User Function TstBloco()

Local bBloco := {|| 2 * 10}
Local nResult
```

```
nResult := EVal(bBloco)          // Poderia ser tambem: EVal( {|| 2 * 10} )

Alert(nResult)

Return Nil

//————————————————————————————————————————————————————————————//
User Function TstBloc3()

Local x
Local y
Local bBloco := {|| x := 10, y := 20}

// Executa uma lista de expressao e retorna o resultado
// da ultima expressao executada.
MsgAlert(EVal(bBloco))

Return Nil

//————————————————————————————————————————————————————————————//
// Bloco de Codigo em forma de string, armazenado numa variavel de tipo caracter,
// e convertido para bloco de codigo por meio de Macro.

User Function TstBloc4()

Local x
Local y
Local cBloco := "{|| x := 10, y := 20}"
Local bBloco

bBloco := &cBloco

MsgAlert(EVal(bBloco))

Return Nil
```

O que você aprendeu neste capítulo

Neste capítulo aprendemos sobre: Macro, lista de expressões e bloco de código.

Próximo Passo

No próximo capítulo, iremos aprender: sobre validação de campo.

Anotações

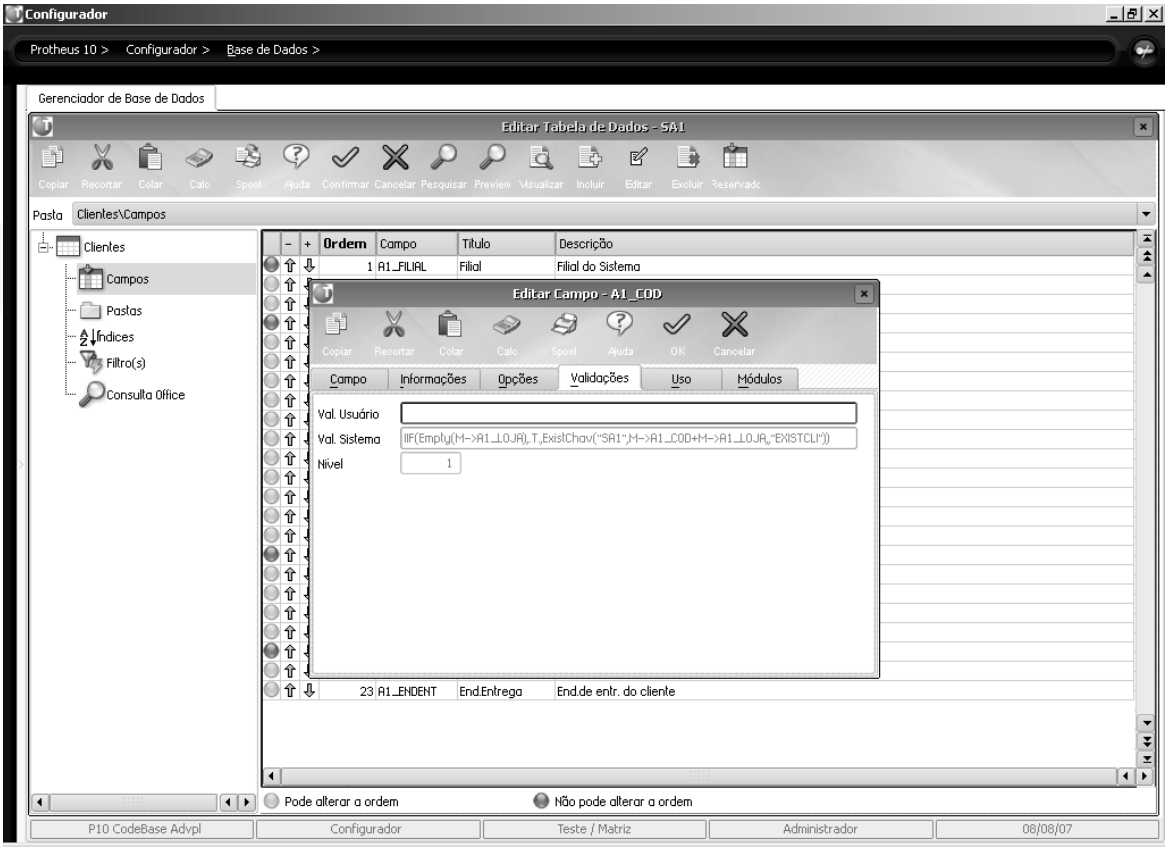
VALIDACÃO DE CAMPO

No ambiente Configurador, menu Base de dados/Dicionário/Bases de dados, é possível ter acesso às configurações dos campos das tabelas, dentre todas as configurações é viável criar mecanismos para críticas de campos no momento da digitação.

Esta funcionalidade ajuda a impedir a inclusão de informações incorretas no banco de dados, bloqueando o seguimento do preenchimento do cadastro, se a função for padrão Microsiga, será enviado um aviso de alerta e a não permissão da informação digitada.

E importante salientar que este mecanismo é semelhante aos mecanismos do gatilho, porém aqui é necessário devolver um valor, sendo verdadeiro (.T.) ou falso (.F.), porque é isso que a validação espera.

Na configuração de validação para campos, também é possível utilizar-se deste recurso no X1_VALID, no qual o objetivo é validar a digitação feita na tela de parâmetros.



Exercício 08 – Criar validação nos campos do cadastro de clientes.

Premissa: Faça uma validação no campo A1_NOME, utilizando a função Texto().

Acesse o ambiente configurador

Selecione a opção: Base de dados/Dicionário/Bases de dados

Clique na opção Dicionário

Pesquise a tabela SA1

Clique em Editar

Selecione a opção Campos

Escolha o campo A1_NOME

Clique em Editar

Selecione a pasta Validação

No campo Validação do Usuário, digite: Texto()
Esta função não permite que o usuário deixe um espaço em branco antes da primeira palavra, ou seja, nenhum nome deverá começar com espaço em branco.

Após a digitação da função, confirme todas as operações, com o objetivo de sair do ambiente configurador.

Acesse o ambiente faturamento e selecione a opção Clientes localizado no menu Atualização/Cadastro, clique na opção incluir e tente informar um nome que comece com espaço em branco, quando terminar tecle <Entre>, perceba que o Sistema criticará a digitação e não permitirá que saia do campo antes de corrigir.

Exercício 09– Criar validação no campo Armazém padrão no cadastro de produtos.

Premissa: Faça uma validação para o campo Armazém padrão no cadastro de produtos para que aceite somente armazém 01 (zero um) e 10 (dez).

Acesse o ambiente configurador

Selecione a opção: Base de dados/Dicionário/Bases de dados

Clique na opção Dicionário

Pesquise a tabela SB1

Clique em Editar

Selecione a opção Campos

Escolha o campo B1_LOCPAD

Clique em Editar

Selecione a pasta Validação

No campo Validação do Usuário, digite: Pertence("01/10")

Esta função permite somente que seja digitada a informação configurada.
Após a digitação da função, confirme todas as operações, com o objetivo de sair do ambiente configurador.

Acesse o ambiente faturamento e selecione a opção Produtos localizado no menu Atualização/Cadastro, clique na opção incluir e tente informar um armazém padrão que não está configurado na função, perceba que o sistema criticará a digitação e não permitirá que saia do campo antes de corrigir.

Exercício 10 – Criar validação nos campos Tipo de entrada e Tipo de saída padrão para o cadastro de produto.

Premissa: os campos B1_TE (tipo de entrada padrão) e B1_TS (tipo de saída padrão), já possuem validação, sendo o campo B1_TE < 500 e no campo B1_TS >= 501, esta definição é padrão do Sistema, então vamos melhorar, nestes campos além desta validação, configure para que está validação permaneça porém envie uma mensagem de alerta ao usuário sobre a ocorrência em cada campo.

Acesse o ambiente configurador

Selecione a opção: Base de dados/Dicionário/Bases de dados

Clique na opção Dicionário

Pesquise a tabela SB1

Clique em Editar

Selecione a opção Campos

Escolha o campo B1_TE

Clique em Editar

Selecione a pasta Validação

No campo Validação do Usuário, digite:

Iif(M->B1_TE<"500",.T.,Eval({|| Help("",1,"F4_TIPO"),.F.}))

Após a digitação da função, confirme todas as operações com o objetivo de sair do ambiente configurador.

Acesse o ambiente faturamento e selecione a opção Produtos localizado no menu Atualização/Cadastro, clique na opção incluir e tente informar um TE padrão que não seja de entrada, perceba que o Sistema criticará a digitação e não permitirá que saia do campo antes de corrigir.

Faça o mesmo procedimento para o campo B1_TS, porém na fórmula digite:

Iif(M->B1_TS >= "501",.T.,Eval({|| Help("",1,"F4_TIPO"),.F.}))

Exercício 11 – Crie uma validação no campo emissão de título a receber.

Premissa: Crie uma validação onde o usuário somente poderá incluir um título no contas a receber quando a emissão for posterior a data de limite para realizações financeira, ou seja, a data definida no parâmetro MV_DATAFIN

Exemplo: MV_DATAFIN igual a 31/01/2004, então se ao incluir um título com data inferior a data definida no parâmetro, o Sistema deve criticá-lo com uma mensagem para o usuário.

Acesse o ambiente configurador

Selecione a opção: Base de dados/Dicionário/Bases de dados

Clique na opção Dicionário

Pesquise a tabela SE1

Clique em Editar

Selecione a opção Campos

Escolha o campo E1_EMISSAO

Clique em Editar

Selecione a pasta Validação

No campo Validação do Usuário, digite:

Iif(M->E1_EMISSAO>=GetMv("MV_DATAFIN"),.T.,Eval({||MsgInfo("Emissão Inválida",""Fim"),.F.}))

Após a digitação da função, confirme todas as operações com o objetivo de sair do ambiente configurador.

Acesse o ambiente financeiro e selecione a opção Contas a Receber localizado no menu Atualização/ Contas a Receber, clique na opção incluir e tente informar uma emissão anterior a data definida no parâmetro MV_DATAFIN, perceba que o Sistema criticará a digitação e não permitirá que saia do campo antes de corrigir.

Note que se mudar a data base do Sistema para uma data anterior a definida no parâmetro MV_DATAFIN, em seguida for incluir um título a receber, o Sistema não fará a mesma crítita, isto porque o campo emissão já inicia com um preenchimento automático, então para se resolver está inconformidade, volte ao ambiente configurador e no campo em questão coloque no seu inicializador padrão a seguinte instrução:

Iif(dDataBase<GetMv("MV_DATAFIN"),Ctod(" / / "),dDataBase), ou seja, se a data base do Sistema for inferior a data definida no parâmetro, este campo inicializará sem preenchimento, obrigando o usuário preenchê-lo.

Anotações

Comandos definidos pelo usuário (UDCs)

UDC USER DEFINED COMMANDS

Os UDCs tem como objetivo principal facilitar a Legibilidade e Manutenibilidade do Fonte

Eles são lidos apenas pelo compilador que gera o objeto de acordo com suas diretivas.

O #DEFINE pode ser usado para permitir que o programador use palavras diferentes daquelas definidas pelo AdvPl e também para definir palavras-chave para o UDC #IFDEF ou #IFNDEF.
Ex: ver TstUDC

As palavras chaves também podem ser definidas no ato de compilar na opção

O IFDEF permite por exemplo que ao se ter um programa que deva atender as condições do Brasil e do México, tenha no mesmo fonte tudo que for comum aos dois e dependendo do #Define BRASIL ou #Define México sejam compilados apenas as partes referentes a cada um dos paises.

O #COMMAND traduz um comando escrito pelo programador de forma simplificada para a forma desejada pelo compilador. Com isso pode-se manter a compatibilidade de fontes escritos em momentos diferentes quando houverem mudanças na sintaxe do AdvPl.

O #INCLUDE indica em que arquivo .CH estão os UDCs a serem usados. Estes por sua vez ficam na pasta INCLUDE. Assim, dependendo da situação altera-se o #INCLUDE e tem-se um resultado diferente da compilação. Por exemplo, compilar para o ambiente Windows ou Linux.

SEMAFOROS: Alguns campos de numeração do Protheus são fornecidos pelo sistema em ordem ascendente. É o caso, por exemplo, do número do Pedido de Venda e outros que servem como chave primária de arquivos. É preciso ter um controle do fornecimento dos números principalmente que vários usuários estão trabalhando simultaneamente.

O conceito do semáforo é o seguinte: ao ser fornecido um número fica ele reservado até a conclusão da digitação da tela. Se confirmada, o número é indisponibilizado para sempre. Se a tela é abandonada o número volta a ficar disponível mesmo que naquele momento números maiores já tenham sido oferecidos. Com isso mesmo que tenhamos vários usuários digitando, por exemplo, Pedidos de Venda, teremos para cada um números exclusivos e nenhum número não utilizado.

Se porem, futuramente um Pedido for cancelado, o seu número não é reaproveitado.

São 4 as funções utilizadas neste processo:

GETSXENUM(alias): obtém o número seqüencial do alias especificado no parâmetro

CONFIRMSXE: confirma o número

ROLLBACKSXE: descarta o número.

MAYIUSE(número): verifica se aquele número pode ser usado. Por exemplo pode-se querer dar um número diferente a um Pedido que está sendo digitado. Mas este número não pode já ter sido usado.

Chaves Primárias

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

Chaves Estrangeiras

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

Anotações

Integridade Referencial

Todos os valores da chave estrangeira têm, obrigatoriamente, que ter valor correspondente na chave primária que se relaciona, mas nem todos os valores encontrados na chave primária, precisam ter seus correspondentes na chave estrangeira que se relaciona.

Por exemplo, na tabela de clientes, o campo A1_COD (código do cliente) vai estar relacionado com outra tabela que indica quais são os pedidos de venda colocados. Desta forma, nem todos os clientes precisam ter pedidos de venda colocados, mas, necessariamente, todos os pedidos de venda precisam de um cliente.

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

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

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

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

Anotações

FUNCÕES APLICADAS EM GATILHOS

O que você irá aprender neste capítulo

Neste capítulo, aprenderemos como criar Funções em ADVPL e utilizá-las através do uso de Gatilhos, ou seja, como disparar um programa customizado através do preeenchimento de um determinado Campo em uma tela de digitação de dados, como por exemplo, um cadastro.

Rotinas Abordadas

• Funções em Gatilhos.

Funções em Gatilhos

A utilização de Gatilhos em cadastros é muito grande, pois através deles poderemos automatizar o preenchimento de muitos Campos durante a digitação de apenas um único Campo de Origem, considerado o Campo de Disparo.

Um Gatilho deve ser utilizado sempre que se fizer necessário a atualização automática de Campos dependentes na tela de digitação (Cadastros ou Movimentações), através do preenchimento de um Campo de Origem.

Dica

• Nunca se esqueça que para se "Compilar um Programa", será necessário a utilização de um "Projeto de Trabalho", sempre;

• Para facilitar a "Visualização", durante a elaboração de programas, não se esqueça de fixar a "Caixa de Projetos" ao rodapé do Protheus IDE;

• Para isto basta clicar sobre o "Cabeçalho da Caixa" e arrastá-lo até o rodapé, juntamente com as pastas de "Mensagens" e "Comandos".

Exercício 12 – Como criar Funções, para utilização em Gatilhos:

1. Selecione as seguintes opções "Arquivo" + "Novo Arquivo";

2. Na "Página de Edição", informe o "Programa" a seguir:

```
#Include "Rwmake.ch"
User Function Exerc01()
c_Nome:="Microsiga"
MsgAlert("Exemplo de Função em Gatilhos.")
Return(c_Nome)
```

3. Selecione as seguintes opções "Arquivo" + "Salvar";

4. Selecione a pasta "C:\xxxxxx", salve o programa como "Exerc01" e confirme;

5. Adicione o "Programa" ao projeto em uso, clicando com o botão direito do mouse sobre a pasta "Fontes" do mesmo, selecionando a opção "Adicionar arquivo", posicionando o cursor sobre o programa e confirmando;

6. Compile-o, clicando com o botão direito do mouse sobre o programa em questão e selecionando a opção "Compilar arquivo...";

7. Será solicitada o "Usuário", no caso, "Administrador" e a "Senha", em branco.

Dica

• Existe na "MICROSIGA", um padrão para a definição da nomenclatura dos "Tipos de Variáveis", como por exemplo:

c_Var	Caracter
n_Var	Numérica
d_Var	Data
l_Var	Lógica
m_Var	Memo
a_Var	Array

• Toda "Função", utilizada em um "Gatilho", sempre deverá ter como "Parâmetro de Retorno", uma "Variável" associada ao programa, para que o "Gatilho" tenha efeito.

Anotações

Significado dos campos da tela de cadastro de gatilhos

Campo – nome do campo que ao ser alterado inicia o processo de atualização.

Seqüência – número seqüencial gerado pelo sistema para cada gatilho.

Cnt. Domínio – nome contra-domínio, ou seja, campo que deverá ser atualizado automaticamente.

Tipo – tipo do gatilho (P/E/X);

P – primário: para atualizações de dados do mesmo arquivo;

E – estrangeiro: para atualizações de dados em outros arquivos;

X – Posicionamento: para posicionar o arquivo mencionado no alias sem efetuar nenhuma atualização. Utilizado para casos em que o usuário deseja estabelecer um relacionamento entre arquivos.

Regra – expressão em sintaxe AdvPL a ser transportada para o contra-domínio, ou apenas o nome de campo. Lembre-se sempre que a regra deve retornar um conteúdo válido conforme o tipo do campo definido como contra domínio.

Posiciona – Sim/Não – movimenta o ponteiro em outro arquivo com base na expressão definida no campo chave.

Alias – Alias do arquivo. As três letras iniciais da tabela cujo o ponteiro deve ser movimentado.

Ordem – número da chave do índice a ser atualizada para movimentação do ponteiro.

Chave – expressão em sintaxe AdvPl que determina o posicionamento do ponteiro.

Condição – informe a condição, user function, etc., que irão determinar quando o gatilho deve ser executado.

Exercício 13 Como criar Gatilhos, para Implementação de Funções:

1. Acesse o "Módulo Configurador";

2. Selecione as seguintes opções "Base de Dados" + "Dicionário" + "Gatilhos";

3. Clique na opção "Incluir" e informe os dados a seguir:

Campo:	A1_END
Sequência:	001
Contra Domínio:	A1_ENDCOB
Tipo	1=Primário
Regra:	EXECBLOCK("EXERC01") ou U_EXERC01()

4. Confira os dados e confirme o cadastro de "Gatilhos".

Dica

• A "Função – Execblock()", é uma "Função Master do Advpl", que executa um "Programa Customizado" que estiver "Compilado no RPO" em uso pelo ambiente e em seguida retorna à execução normal do sistema;

• Sempre poderemos utilizar a chamada a uma "Função" das duas maneiras descritas acima, ou seja, através da "Função – Execblock()" ou diretamente pelo nome da "User Function", que está compilada no RPO em uso.

Exercício 14 – Como Testar o Gatilho:

1. Acesse o "Módulo Financeiro";

2. Selecione as seguintes opções "Atualização" + "Cadastro" + "Clientes";

3. Clique na opção "Incluir";

4. No "Campo – Endereço", informe um "Endereço" à sua escolha e pressione a tecla <Enter>;

Observação: Note que a "Mensagem de Aviso" será disparada, isso demonstra que o "Gatilho" foi disparado.

5. Em seguida, clique na pasta "Adm/Fin." e verifique o "Resultado do Gatilho" no "Campo – End. Cobrança", que deverá estar preenchido com "Microsiga".

Exercício 15– Como criar Funções, para retornar a Variável em Memória do Campo:

1. Altere o "Programa – Exerc01" com as seguintes informações:

```
#Include "Rwmake.ch"
User Function Exerc01()
c_Nome:=M->A1_END
MsgAlert("Exemplo de Função em Gatilhos.")
Return(c_Nome)
```

2. Selecione as seguintes opções "Arquivo" + "Salvar" e compile-o novamente;

3. Realize o "Teste", acessando novamente o "Módulo Financeiro" e executando o "Cadastro de Cliente", como no exemplo anterior.

Dica

Lembre-se que quando a informação está sendo digitada, a mesma encontra-se no "Buffer", então devemos tratar como "M-><campo>", que identifica que a informação de retorno está na memória e quando a informação já está gravada, deveremos então tratar como "<Alias>-><Campo>".

Exercício 16 – Como utilizar Funções com Retorno a partir de uma determinada Condição de Preenchimento do Campo:

• Crie um programa com o nome de "Exerc02", que deverá atualizar o "Campo – Bairro" do "Cadastro de Clientes", para "Centro", quando o "Campo – CEP", for igual a "00000-000".

1. Digite o programa a seguir:

```
#Include "Rwmake.ch"
User Function Exerc02()
If M->A1_CEP=="00000000"
        c_Bairro:="Centro"
Else
        c_Bairro:=M->A1_BAIRRO
Endif
Return(c_Bairro)
```

2. Salve o "Programa", adicione-o ao "Projeto" e "Compile-o".

Exercício 17 – Como criar o Gatilho:

Agora, crie o "Gatilho", para a "Função" acima, da seguinte maneira:

Campo:	A1_CEP
Sequência:	001
Contra Domínio:	A1_BAIRRO
Tipo	1=Primário
Regra:	U_EXERC02()

1. Acesse o "Módulo Financeiro";

2. Selecione as seguintes opções "Atualizações" + "Cadastros" + "Clientes";

3. Clique na opção "Incluir" e informe "00000000" no "Campo – CEP" e observe que o retorno no "Campo – Bairro", foi disparado.

Exercício 18 – Como criar Funções, utilizando mais de uma Condição de Disparo:

• Crie um programa com o nome de "Exerc03", que deverá atualizar o "Campo – Valor do Título" do cadastro de Contas a Receber, com "10%" a mais sobre o "Valor Digitado", quando o "Campo – Tipo", for igual a "DP", o "Campo – Cliente", for igual a "000001" e o "Campo – Loja", for igual a "01".

1. Digite o programa a seguir:

```
#Include "Rwmake.ch"
User Function Exerc03()
If  M->E1_TIPO="DP".and.M->E1_CLIENTE="000001".and.M->E1_LOJA="01"
        n_Valor:=M->E1_VALOR*1.1
Else
        n_Valor:=M->E1_VALOR
Endif
Return(n_Valor)
```

2. Salve o "Programa", adicione-o ao "Projeto" e "Compile-o";

Exercício 19 – Como Criar o Gatilho:

1. Crie o seguinte "Gatilho", para a "Função" acima:

Campo:	E1_VALOR
Sequência:	011
Contra Domínio:	E1_VALOR
Tipo	1=Primário
Regra:	U_EXERC03()

Exercício 20 – Como Testar o Gatilho:

1. Acesse o "Módulo Financeiro";

2. Selecione as seguintes opções "Atualizações" + "Contas a Receber" + "Contas a Receber";

3. Clique na opção "Incluir", preencha o "Campo – Tipo" com "DP", no "Campo – Cliente", pressione a tecla <F3> e informe os dados a seguir:

4. Selecione-o e continue o cadastro, preenchendo o "Campo – Vlr. Título" com "100,00" e verifique que o mesmo será alterado com "10%" a mais sobre o "Valor Digitado".

Código:	000001
Loja:	01
Nome:	Cliente 01
N Fantasia:	Cliente 01
Tipo:	R=Revendedor
Endereço:	Avenida Braz Leme, 1631
Município:	São Paulo
Estado:	SP

Exercício 21 – Como criar Funções utilizando Matriz:

• Crie um programa com o nome de "Exerc04", que deverá atualizar o "Campo – DDD" do "Cadastro de Clientes", dependendo da "Unidade Federativa", que for preenchida no "Campo – Estado".

1. Digite o programa a seguir:

```
#Include "Rwmake.ch"
User Function Exerc04()
Local aFone := {}
Local nP := 0
Local cRet := ""
aAdd( aFone, {"SP","11"} )
aAdd( aFone, {"RJ","21"} )
aAdd( aFone, {"MG","31"} )
aAdd( aFone, {"AM","91"} )
aAdd( aFone, {"BA","71"} )
nP := aScan( aFone, {|x|x[1]==M->A1_EST})
If nP <> 0
        cRet := aFone[nP][2]
Endif
Return( cRet )
```

2. Salve o "Programa", adicione-o ao "Projeto" e "Compile-o";

Exercício 22 – Como Criar o Gatilho:

1. Crie o "Gatilho" a seguir, para a "Função" acima:

Campo:	A1_EST
Sequência:	001
Contra Domínio:	A1_DDD
Tipo	1=Primário
Regra:	U_EXERC04()

Exercício 23 – Como Testar o Gatilho:

1. Acesse o "Módulo Financeiro";

2. Selecione as seguintes opções "Atualizações" + "Cadastros" + "Clientes";

3. Clique na opção "Incluir" e preencha o "Campo – Estado", com "SP";

4. Verfique que o "Campo – DDD", será atualizado de acordo com o retorno da "Matriz", utilizada na "Função" disparada pelo "Gatilho".

Exercício 24 – Como criar gatilho para atualizar campo que depende da informação de outra tabela.

Premissa: no cadastro de complemento de produtos, é necessário digitar o código do produto. Faça um gatilho que após está digitação o campo nome cientifico seja preenchido automaticamente com o nome do produto, conforme o código selecionado.

Campo	B5_COD
Seqüência	001
Contra Domínio	B5_CEME
Tipo	1=Primário
Regra	SB1->B1_DESC
Posicione	Sim
Alias	SB1
Ordem	1
Chave	xFilial("SB1")+M->B5_COD

Após a inclusão do gatilho, teste-o acessando o ambiente Estoque/Custos. No menu, selecione a opção Complemento de Produtos, localizado no menu Atualizações/Cadastro.

Exercício 25-Utilizando o exercício anterior, faça que este Gatilho só funcione no Módulo Estoque. Apenas preencha o campo condição no cadastro do gatilho anterior criado.

Condição Cmodulo=="EST"

Para testar este exercício, acesse o ambiente Faturamento e inclua um complemento de produto.

Repare que a ação do gatilho não funcionará, devido a condição colocada ao gatilho configurado. Em contra partida, se for incluir um complemento de produto pelo ambiente Estoque/Custo, ele terá sua ação funcionando normalmente.

O que você aprendeu neste capítulo

Neste capítulo aprendemos a criar Gatilhos com utilização de Funções.

Próximo Passo

No próximo capítulo, iremos aprender como criar Telas de Cadastros utilizando as Funções da Microsiga e também à desenvolver Programas de Validação e Relacionamento entre arquivos.

Anotações

FUNCÕES PARA CADASTROS

O que você irá aprender neste capítulo

Neste capítulo, aprenderemos como criar Telas para Cadastros, utilizando para isso as Funções definidas pela MICROSIGA.

Veremos também, Funções Customizadas, para Validações em Arquivos e Relacionamentos.

Rotinas Abordadas

• Funções para Cadastros:
  – AxCadastro();
  – MBrowse();
  – Modelo2();
  – Modelo3().

Desenvolvendo Telas

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

O MP8 já cria, automaticamente, a grande parte das telas de um ambiente, tais como a Browse, a GetDados e Enchoice. Algumas outras telas necessitam de construção "manual", ou seja, com a utilização de comandos, tais como "SAY", "GET" e "LABEL" na Dialog. Procure sempre colocar em tela as informações que mais se objetivam com o assunto abordado.

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

Quando o volume de informações é muito grande, divida os campos em folders, ou seja, pastas, agrupando os campos em assuntos. Isso irá deixar a tela menos poluída e evitará que o usuário navegue por uma tela só.

Para fazer essa facilidade, preencha o campo X3_FOLDER no SX3, com um número, agrupando-os de acordo com o tipo de informação e no SXA, com o ALIAS do arquivo em pauta, a ordem, que equivale ao número informado no X3_FOLDER e a descrição nos três idiomas.

Essa descrição que será a informação contida na pasta do folder. Exemplo: Os campos SZ1_ENDER, SZ1_NUM e SZ1_BAIRRO devem estar com o campo X3_FOLDER preenchido com o conteúdo "1".

No SXA, o XA_ALIAS deverá ser SZ1, o XA_ORDEM = "1" (mesmo valor preenchido no X3_FOLDER), no XA_DESCRIC, "Endereço Residencial" e, nos demais, o mesmo texto em outros idiomas.

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

Evite tela com muitos botões. Isso poderá confundir o usuário e induzi-lo ao erro. Utilize telas seqüenciais, conhecidas como Wizard (semelhante aos de instalação de um software). Dessa forma, o usuário ficará mais atento aos fatos, dificultando o erro. Mas, cuidado: não faça disso uma incansável seqüência de telas, pois isso acabará desmotivando o usuário a utilizar o Sistema.

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

Exercício 26 – Utilizando os objetos para criar telas.

```
#Include "PROTHEUS.CH"

User Function Tela1()

Local oDlg
Local oBtnOk
Local oBtnCancel
Local aButtons
Local oSayConj
Local oGetConj
Local oSayCivil, cSayCivil
Local oSaySal, cSaySal
Local oRadio, nRadio
Local oChk, lChk
Local cNome    := Space(20)
Local cConjuge := Space(20)
Local cEnder   := Space(30)
Local cCivil   := Space(1)
Local oFont
Local oFolder

Define Font oFont Name "Arial" Size 0,-12 Bold

aButtons := {{"BMPPERG",{||MsgInfo("Pergunte")},"Pergunte..."},;
        {"BMPCALEN",{||MsgInfo("Calendario")},"Calendario..."}}

Define MSDialog oDlg Title "Exemplo " From 0,0 To 500,600 Pixel

@20,10 Say "Nome:" Pixel Of oDlg
@20,50 Get cNome Size 50,10 Pixel Of oDlg

@40,10 Say "Estado Civil:" Pixel Of oDlg
@40,50 Get cCivil Size 10,10 Picture "@!" Valid cCivil$"S|C|D" .And. u_VldCivil(cCivil, oSayConj, oGetConj,
oSayCivil) Pixel Of oDlg

@40,80 Say oSayCivil Var cSayCivil Size 20,10 Pixel Of oDlg

@60,10 Say oSayConj Var "Cônjuge:" Pixel Of oDlg
@60,50 Get oGetConj Var cConjuge Size 50,10 Pixel Of oDlg
```

```
@80,10 Say "Endereço:" Pixel Of oDlg
@80,50 Get cEnder Pixel Of oDlg

@100,10 Say "Salário:" Pixel Font oFont Of oDlg

@100,40 Radio oRadio Var nRadio Items "1000", "2000", "3000" Size 50,9 On Change u_Salario(nRadio, oSaySal) Pixel Of oDlg

@100,80 Say oSaySal Var cSaySal Size 20,10 Pixel Of oDlg

@140,10 CheckBox oChk Var lChk Prompt "Check Box" Size 70,9 On Change MsgAlert(If(lChk,"Marcado","Desmarcado")) Pixel Of oDlg

@oDlg:nHeight/2-30,oDlg:nClientWidth/2-70 Button oBtnOk    Prompt "&Ok"    Size 30,15 Pixel Action u_Confirma()    Message "Clique aqui para Confirmar" Of oDlg
@oDlg:nHeight/2-30,oDlg:nClientWidth/2-35 Button oBtnCancel Prompt "&Cancelar" Size 30,15 Pixel Action oDlg:End() Cancel Message "Clique aqui para Cancelar"  Of oDlg

@160,10 Folder oFolder Prompts "Pasta 1","Pasta 2" Size 200,80 Pixel Of oDlg
oFolder:aDialogs[2]:oFont := oFont
//oFolder:bChange := {||MsgAlert("Mudando de pasta")}

@10,10 Say "Conteudo da Primeira Pasta" Pixel Of oFolder:aDialogs[1]
@10,10 Say "Conteudo da Segunda Pasta"  Pixel Of oFolder:aDialogs[2]

Activate MSDialog oDlg Centered On Init EnchoiceBar(oDlg, {||u_OK(),oDlg:End()}, {||oDlg:End()},,aButtons)

Return Nil

//————————————————————————————————————————————————————————————————————//
User Function Confirma()

MsgAlert("Você clicou no botão OK!")

// Aqui poderia, por exemplo, gravar os dados num arquivo.

Return

//————————————————————————————————————————————————————————————————————//
User Function OK()

MsgAlert("Você clicou no botão OK da EnchoiceBar!")

// Aqui poderia, por exemplo, gravar os dados num arquivo.

Return

//————————————————————————————————————————————————————————————————————//
```



```
User Function VldCivil(cCivil, oSayConj, oGetConj, oSayCivil)

If cCivil <> "C"
  oSayConj:Hide()
  oGetConj:Hide()
  //oSayConj:Disable()
  //oGetConj:Disable()
 Else
  oSayConj:Show()
  oGetConj:Show()
  //oSayConj:Enable()
  //oGetConj:Enable()
EndIf

If      cCivil == "C"
        oSayCivil:SetText("Casado")
 ElseIf  cCivil == "S"
        oSayCivil:SetText("Solteiro")
 Else
        oSayCivil:SetText("Divorciado")
EndIf

Return .T.

//————————————————————————————————————————————————————————————
—//
User Function Salario(nRadio, oSaySal)

If nRadio == 1
  oSaySal:SetText("Hum mil")
 ElseIf nRadio == 2
  oSaySal:SetText("Dois mil")
 Else
  oSaySal:SetText("Tres mil")
EndIf

Return
```

Podemos dizer que a criação da tela se divide em duas partes: Definição e Ativação.

Dica

Na linha Define MsdiAlog oDlg Title : "Exemplo" From 000,000 to 400,600 pixel. Esta acontecendo a definição da tela. Interpretando o código: Define MsDialog oDlg: estamos definindo uma janela. E o nome objeto responsável pela criação da mesma é o oDlg. Este, simplesmente;é um nome de variável. Poderia ser oJanela, por exemplo. Title : "Exemplo". Esta atribuindo como título da janela a expressão : "exemplo". From 000,000 to 400,600 pixel. Esta definindo as coordenadas da janela.

Na linha Active MsDialog. Esta ativando a janela. Indica que vai ativar o objeto oDlg centralizado na tela.
On init Enchoicebar. Este trecho significa que: quando a janela for iniciada será criada uma Enchoicebar (barra de ferramentas).

AxCadastro() – Cadastro Padrão

Esta Função é utilizada para a Criação de Telas de cadastros padronizados da MICROSIGA, onde poderemos ter apenas as opções de "Pesquisar", "Visualizar", "Alterar", "Incluir" e "Excluir", sendo que os mesmo não poderão ser modificados.

Sintaxe:

AxCadastro(cAlias,cTitulo,cDel,cOk);

Parâmetros:

cAlias = Alias do Arquivo para o cadastro, deve obrigatoriamente ter a sua Estrutura definida no SX3;
cTitulo = Título da Janela;
cDel = Função para validar a Exclusão;
cOK = Nome de Função para Validação da Tela.

Exercício 27 – Como criar Telas de Cadastros, com utilização de Arquivo Padrão:

• Crie uma "Tela de Cadastro Padrão", para o arquivo de "Cadastro de Produtos" e adicione-o ao menu do Módulo de Faturamento. Utilize para isso um programa com o nome de "Exerc05".

1. Crie o "Programa" a seguir:

```
#Include "Rwmake.ch"
User Function Exerc05()
Axcadastro("SB1","Exerc05")
Return
```

Observação:

Salve o "Programa", adicione ao "Projeto" e "Compile-o", adicione Programas ao Menu acessando o "Módulo Configurador".

Lembre-se de que, "Jamais", deveremos utilizar um "Menu Original", para nossas customizações, o correto é gerarmos uma "Cópia", para que não se perca as configurações do menu original.

Exercício 28 – Como Testar o Cadastro:

1. Acesse o "Módulo de Faturamento";

2. Selecione as seguintes opções "Atualização" + "Cadastros" + "Exerc05";

3. Clique na opção "Incluir" e cadastre os "Produtos" a seguir:

Código:	100.100-01
Descrição:	CANETA AZUL
Tipo:	MC
Unidade:	PC
Armazem Pad:	01

Código:	100.200-01
Descrição:	CANETA VERMELHA
Tipo:	MC
Unidade:	PC
Armazem Pad:	01

4.Confira os dados, confirme e saia do "Cadastro de Produtos".

Exercício 29 – Como cadastrar a Previsão de Vendas:

1. Selecione as seguintes opções "Atualização" + "Cenários de Venda" + "Previsão de Vendas";

2. Clique na opção "Incluir" e informe as "Previsões" a seguir:

Produto:	100.100-01
Almoxarifado:	1
Quantidade:	1,00

Produto:	100.200-01
Almoxarifado:	1
Quantidade:	1,00

3. Confira os dados e confirme o "Cadastro de Previsão de Vendas".

Exercício 30 – Como Verificar a Integridade Relacional entre os dois Cadastros:

1. Selecione as seguintes opções "Atualizações" + "Cadastro" + "Produtos";

2. Posicione com o cursor sobre o "Produto – 100.100-01";

3. Clique a opção "Excluir";

Observação:

Note que o sistema não irá realizar a "Exclusão", devido a amarração deste "Produto" ao cadastro de "Previsão de Vendas".

4. Saia deste "Cadastro" e selecione as seguintes opções "Atualizações" + "Cadastro" + "Exerc05";

5. Posicione com o cursor sobre o "Produto – 100.100-01";

6. Clique na opção "Excluir".

Anotações

Observação:

• Observe que desta vez o sistema realizou a "Exclusão";

• A "Exclusão do Registro" foi possível devido ao fato de que ainda não há nenhuma "Validação", para verificar a "Integridade Relacional" entre os dois arquivos no nosso programa.

Exercício 31 – Como criar Programas, para verificar a Exclusão de Produtos:

• Crie um programa para verificar se o registro selecionado para "Exclusão" no "Cadastro – Exerc05", poderá ou não ser "Excluído". Para isso, será necessário verificar se o mesmo não se encontra relacionado no cadastro de "Previsão de Vendas".

1. Crie o "Programa" a seguir com o nome de "Vdel":

```
#Include "Rwmake.ch"
User Function Vdel()
Local c_Area := Alias(), n_Rec:=Recno(), c_ind:=IndexOrd()
Local c_Cod:=SB1->B1_COD, l_Del:=.T.
Msgbox("Verificando amarracao do produto" + c_Cod,"Atencao!!!")
Dbselectarea("SC4")
If Dbseek(Xfilial("SC4")+c_Cod)
        Msgbox("Produto nao pode ser excluido!!!")
        l_Del:=.F.
Endif
DbSelectArea(c_Area)
DbSetOrder(c_ind)
DbGoTo(n_Rec)
Return(l_Del)
```

2. Salve o "Programa", adicione ao "Projeto" e "Compile-o".

Exercício 32 – Como Alterar o Programa, para Validação da Exclusão:

1. Faça as seguintes modificações no "Programa – Exerc05":

AxCadastro("SB1","Exerc05","Execblock('Vdel')","F.")

2. Salve o "Programa" e "Compile-o", novamente.

Exercício 33 – Como Testar Validações de Exclusões:

1. Acesse o "Módulo de Faturamento";

2. Selecione as seguintes opções "Atualização" + "Cadastros" + "Exerc05";

3. Posicione com o cursor sobre o "Produto –100.200-01";

4. Clique na opção "Excluir";

Observação:

Verifique que a "Exclusão", não será realizada, pois agora existe uma "Validação", para essa rotina;

5. Saia do cadastro de "Produtos";

6. Selecione as seguintes opções "Atualizações" + "Cenários de Venda" + "Previsão de Vendas";

7. Posicione com o cursor sobre a "Previsão – 100.200-01", clique na opção "Excluir" e confirme;
8. Retorne ao "Cadastro – Exerc05" e execute novamente a "Rotina de Exclusão".

Verifique que desta vez a rotina será executada, pois não existe mais nenhuma amarração deste registro com o cadastro de "Previsão de Vendas".

Não poderemos mais incluir registros através da opção "Exerc05", pois o mesmo foi modificado, para retornar "Falso" no parâmetro responsável pela inclusão.

Exercício 34 – Como criar Programas de Exclusão de Registros em Sequência:

• Crie um "Programa" que durante a "Exclusão" de um registro no "Exerc05" seja verificado se o mesmo está amarrado ao cadastro de "Previsão de Vendas", caso esteja, deverá ocorrer a "Exclusão" deste registro em ambos os cadastros.

1. Realize as seguintes modificações no "Programa – Vdel":

```
#Include "RwMake.CH"
User Function vDel()
//Local c_Area := Alias(), n_Rec:=Recno(), c_ind:=IndexOrd()
Local aArea := GetArea() //Substitui a linha acima
Local cCod := SB1->B1_COD, lDel:=.T.
MsgBox("Verificando amarracao do produto " + cCod,"Atencao!!!")
DbSelectArea("SC4")
If DbSeek(xFilial("SC4")+cCod) // Verifica se existe Previsão de Vendas
        If MsgBox("Deseja excluir a Previsão de Vendas","Previsão de Vendas","YESNO")
                lDel := .T.
                If RecLock("SC4")
                        DbDelete()
                Else
                        MsgBox("Registro em uso, o produto não será excluido!")
                        lDel := .F.
                EndIf
        Else
                MsgBox("Produto não será excluído!")
                lDel := .F.
        EndIf
EndIf
RestArea(aArea) // Substitui as 3 linhas acima
Return(lDel)
```

2. Salve o "Programa" e "Compile-o", novamente.

Exercício 35 – Como Testar a Validação, para Exclusão de Registros em Sequência:

1. Acesse o "Módulo de Faturamento";

2. Selecione as seguintes opções "Atualização" + "Cadastros" + "Produtos";

3. Clique na opção "Incluir" e informe os dados a seguir:

Código:	100.100-01
Descrição:	CANETA AZUL
Tipo:	MC
Unidade:	PC
Armazem Pad:	01

4. Confira os dados, confirme e saia do cadastro de "Produtos";

5. Selecione as seguintes opções "Atualização" + "Cenários de Venda" + "Previsão de Vendas";

6. Clique na opção "Incluir" e informe os dados a seguir:

Produto:	100.100-01
Almoxarifado:	1
Quantidade:	1,00

7. Confira os dados e confirme o cadastro de "Previsão de Vendas".

Exercício 36 – Como Testar a Exclusão de Registros em Sequência:

1. Selecione as seguintes opções "Atualização" + "Cadastros" + "Exerc05";

2. Posicione com o cursor sobre o "Produto – 100.100-01";

3. Clique na opção "Excluir" e saia do cadastro de "Produtos";

4. Selecione as seguintes opções "Atualizações" + "Cenários de Venda" + "Previsão de Vendas";

5. Verifique que a "Previsão de Vendas", cadastrada para o "Produto – 100.100-01", também foi "Excluída".

Observação:

Todos os detalhes referentes "Criação de Campos e Manutenção das Estruturas de um Arquivo" são vistos com maiores detalhes no curso do "Módulo Configurador".

• Caso queira criar mais campos, o procedimento é o mesmo, e se desejar "Criar Campos" que tenham, as características semelhantes a de outro campo, basta verificar, como está definida a "Estrutura do Campo", que se deseja "Copiar" e criá-la onde desejar;

Dica

• Sempre que confirmamos a "Alteração da Estrutura de um Arquivo", o Protheus, gera um "Backup", automático do "Dicionário de Dados", como no exemplo a seguir:

SX3990.#DB (Nas versões "ADS" e "Codebase"), ou
SX3990.BKP (Nas versões "SQL").
MBrowse()

Esta Função é utilizada para a criação de telas de cadastros padronizados da MICROSIGA, onde poderemos criar até 10 botões, sem a necessidade de utilizarmos os originais.

Também é possível a utilização de Legendas durante o cadastro, sendo que as mesmas poderão ser definidas através de Cores e Condições.

Sintaxe:

MBrowse(nT,nL,nB,nR,cAlias,aFixe,cCpo,nPosI,cFun,nDefault, aColors,cTopFun,cBotFun)

Parâmetros:

nT = Linha Inicial;
nL = Coluna Inicial;
nB = Linha Final;
nR = Coluna Final;
cAlias = Alias do Arquivo;
aFixe = Array, contendo os Campos Fixos (A serem mostrados em primeiro lugar no Browse);
cCpo = Campo a ser tratado, quando Empty, para informações de Linhas com Cores;

Prioridade 2

nPosI = (Dummy);
cFun = Função para habilitar semáforo; Prioridade 3 na execução; Prioridade 3
nDefault = Número da opção do menu a ser executada quando o <Enter> for pressionado;
aColors = Array com os Objetos e Cores; Prioridade 1
cTopFun = Função para filtrar o TOP do Browse;
cBotFun = Função para filtrar o BOTTOM do Browse.

Observação:

Lembre-se de utlizar o IDE, para criar, compilar e debugar o seu programa.

Exercício 37 – Criando uma Janela com a função MBROWSE:

```
User Function Exerc06
Private aRotina :=      {{"Pesquisar","AxPesqui",0,1} ,;
                        {"Visualizar","AxVisual",0,2} ,;
                        {"Incluir","AxInclui",0,3} ,;
                        {"Alterar","AxAltera",0,4} ,;
                        {"Excluir","AxDeleta",0,5},;
                        {"Compl. Produtos","Mata180()",0,6}}
```

Confira os dados, salve o "Programa" também com o nome de "Exerc06", adicione ao "Projeto" e "Compile-o".14. Adicione-o ao menu do "Modulo Faturamento" e teste.

Exercício 38 – Como utilizar Legendas nos Cadastros:

• Crie um "Programa", para o "Cadastro de Produtos", sendo que o mesmo deverá possuir uma "Legenda" ao lado de cada "Registro" informando se o "Campo – Preço de Venda", está ou não "Preenchido".

1. Realize a seguinte alteração no "Programa – Exerc06":

mBrowse(6,1,22,75,cString,,"B1_PRV1")

2. Confira os dados, salve o "Programa" e "Compile-o" novamente.

3. Insira os dados a seguir:

Código:	000000000000001
Descrição:	PRODUTO 1
Tipo:	PA
Unidade:	CX
Armazém Pad.:	01
Preço Venda:	1.00

Código:	000000000000002
Descrição:	PRODUTO 2
Tipo:	PA
Unidade:	CX
Armazém Pad.:	01

4. Cancele a "Próxima Inclusão";

5. Verifique que o produto que atende à "Condição" estipulada pelo programa está com uma legenda "Vermelha" e a outra "Verde".

Observação:

As "Cores Padrões" das "Legendas", serão sempre "Vermelha" quando a "Condição", for "Satisfeita" e "Verde", quando ainda estiver "Pendente".

Exercício 39 – Como utilizar Legendas com Condições de Preenchimento dos Campos:

• Crie um "Programa", para o "Cadastro de Produtos", sendo que o mesmo deverá possuir uma "Legenda" ao lado de cada registro informando se o "Campo – Preço de Venda" está ou não preenchido com "Valor Igual ou Superior a R$ 100,00".

1. Realize a seguinte alteração no "Programa – Exerc06":

mBrowse(6,1,22,75,cString,,"B1_PRV1>=100")

2. Confira os dados, salve o "Programa" e "Compile-o" novamente e teste.

Observação:

Verifique que a "Cor" do primeiro registro, foi alterada para "Verde", pois a "Nova Condição" ainda não foi "Satisfeita";

3. Posicione com o cursor sobre esse mesmo "Registro" e clique na opção "Alterar";

4. Preencha o "Campo – Preço Venda" com um "Valor Igual ou Superior a R$ 100,00" e confirme;

5. Verifique que agora a "Legenda", mudou novamente para "Vermelho", pois a "Nova Condição", foi "Atendida".

Exercício 40 – Como criar Outra Tela de Cadastros:

• Crie um "Programa" denominado "Exerc601", utilizando o arquivo de "Cadastro de Clientes", onde o mesmo deverá possuir uma "Legenda", informando se o "Campo – Tipo", está preenchido como "Revendedor".

1. Acesse a "Ferramenta – IDE";

2. Selecione as seguintes opções "Ferramentas" + "Assistente de Código";

3. Marque a opção "Cadastro";

4. Clique na opção "Avançar>>";

5. Marque a opção "Arquivo Padrão";

6. No "Campo – Alias", informe o "Arquivo – SA1";

7. Marque a opção "Utiliza índice padrão";

8. Clique na opção "Avançar>>" + "Avançar>>";

9. Marque a opção "MBrowse";

10. Na pasta "Configurações", no "Campo – Título", informe "Cadastro de Clientes";

11. Clique na opção "Finalizar";

12. Realize as seguintes correções no "Programa":

```
User Function Exerc601
mBrowse( 6,1,22,75,cString,,"A1_TIPO== 'R'")
```

13. Confira os dados, salve o "Programa", também com o nome de "Exerc601", adicione ao "Projeto" e "Compile-o".

14. Adicione-o ao menu do "Modulo Faturamento" e teste.

Anotações

Observação:

Verifique que o "Registro" existente, possui a legenda em "Vermelho", pois a "Nova Condição", já se encontra "Satisfeita";

15. Clique na opção "Alterar", preencha o "Campo – Tipo", com "X=Exportação" e confirme;

16. Agora, verifique que a "Legenda", passou para a cor "Verde", confirmando que a "Condição" não foi "Satisfeita", dessa vez e saia do "Cadastro".

Exercício 41– Como Criar Cadastros com Filtros:

• Crie um "Programa" denominado "Exerc602", utilizando o "Arquivo de Tabelas" do "Módulo Configurador", onde o mesmo deverá apresentar, apenas os "Os Estados da Região Sudeste", quando o usuário acessar o sistema, porém deverá apresentar todos, caso o sistema seja acessado pelo "Administrador".

1. Acesse a "Ferramenta – IDE";

2. Crie o "Programa" a seguir:

```
#Include "Rwmake.ch"
User Function Exerc602()
Dbselectarea("SX5")
Dbsetorder(1)
cCadastro:="Tabelas"
aRotina:={{ "Pesquisar","" Axpesqui",0,1},;
        {"Alterar","" AxAltera",0,4}}
If Upper(Substr(cUsuario,7,15))<>"ADMINISTRADOR"
        Set Filter To X5_TABELA == "12" .And. Alltrim(X5_CHAVE) $ "SP/RJ/MG/ES"
Endif
Mbrowse(6,1,22,75,"SX5")
Set Filter to
Return
```

3. Confira os dados, salve o "Programa", com o nome de "Exerc602", adicione ao "Projeto" e "Compile-o".

4. Adicione-o ao menu do "Modulo Faturamento" e teste.

Observação:

Verifique que apenas os "Estados da Região Sudeste", foram exibidos.

5. Saia do "Sistema" e acesse-o novamente como "Administrador", repetindo o "Teste";

6. Verifique que todos os "Estados" serão exibidos e saia do "Cadastro".

Anotações

Exercício 42– Criando Legenda para o "Semaforo" e ativando o mesmo com a tecla <Enter>

• Crie um "Programa" denominado "EXERC13", utilizando o "Cadastro de Produtos" do "Módulo Faturamento", para apresentar uma "Legenda" do "Semaforo", este deverá chamar o Programa de "Legenda" quando pressionado o <Enter>, utilizaremos neste exercício os parâmetros "cDefault" e "aCores".

1. Acesse a ferramenta IDE;

2. Crie o programa a seguir;

```
#INCLUDE "rwmake.ch"
User Function Exerc13()
Private cCadastro := "Cadastro de Produtos"
Private aRotina := { {"Pesquisar","AxPesqui",0,1} ,;
                     {"Visualizar","AxVisual",0,2} ,;
                     {"Incluir","AxInclui",0,3} ,;
                     {"Alterar","AxAltera",0,4} ,;
                     {"Excluir","AxDeleta",0,5} ,;
                     {"Legenda","u_Legenda()",0,6} }
Private cDelFunc := ".T." // Validacao para a exclusao. Pode-se utilizar ExecBlock
aCores := {{"B1_PRV1 == 0","BR_VERMELHO"},;
          {"B1_PRV1 > 0 .And. B1_PRV1 <= 100","BR_LARANJA"},;
          {"B1_PRV1 > 100 .And. B1_PRV1 <= 200","BR_AZUL"},;
          {"B1_PRV1 > 200","BR_VERDE"}}
dbSelectArea("SB1")
dbSetOrder(1)
mBrowse( 6,1,22,75,"SB1",,,,,6,aCores)
Return
User Function Legenda()
BrwLegenda(cCadastro,"Valores",{    {"BR_VERMELHO","Preço não informado"},;
                                    {"BR_LARANJA","Preço > 0 e <= 100" },;
                                    {"BR_AZUL","Preço > 100 e <= 200" },;
                                    {"BR_VERDE","Preço > 200" }}    )
Return .T.
```

3. Confira os dados, salve o programa "EXERC13", adicione ao "Projeto" e "Compile-o".

4. Adicione ao menu do "Modulo Faturamento"

Exercício 43 – Como criar telas de cadastros pela função Mbrowse e fazendo a chamda pela função MODELO2.

Observação:

Lembre-se de utlizar o IDE, para criar, compilar e debugar o seu programa.
Tudo que esta entre /* e */ ou // é comentário.

Altere somente o que estiver em Negrito

```
#include "Protheus.ch"
User Function Exerc08()
```

```
//————————————————————————————————————————————————————————+
//| Declarar as variaveis aRotina e cCadastro como Private, porque a função mBrowse está esperando |
//————————————————————————————————————————————————————————+

Private aRotina := {}
Private cCadastro := "Solicitação de Software"

//————————————————————————————————————————————————————————+
//| Montar o vetor aRotina, obrigatorio para a utilização da funçã mBrowse() |
//————————————————————————————————————————————————————————+

aAdd( aRotina,{"Pesquisar" ,"AxPesqui"   , 0 , 1 })
aAdd( aRotina,{"Visualizar","u_Mod2Manut", 0 , 2 })
aAdd( aRotina,{"Incluir"   ,"u_Mod2Manut", 0 , 3 })
aAdd( aRotina,{"Alterar"   ,"u_Mod2Manut", 0 , 4 })
aAdd( aRotina,{"Excluir"   ,"u_Mod2Manut", 0 , 5 })

//————————————————————————————————————————————————————————+
//| Selecionar a tabela ordem e posicionar no primeiro registro da ordem |
//————————————————————————————————————————————————————————+

dbSelectArea("SZ4")
dbSetOrder(1)
dbGoTop()

//————————————————————————————————————————+
//| Executar a função mBrowse para a tabela mencionada |
//————————————————————————————————————————+

mBrowse(,,,,"SZ4")
Return
```

Exercício 44 – Criado a tela de inclusão com a função Modelo2.

Anotações

Sintaxe da Função Modelo2()

Esta Função é utilizada para a Criação de Telas de Cadastros onde poderemos trabalhar com um Esquema Cabeçalho desenvolvido apenas com a utilização de Variáveis de Memória e logo abaixo uma tela com linhas, utilizada para a digitação dos dados, aparentando estarmos trabalhando com dois arquivos em tela, porém na verdade existirá apenas um.

Como exemplo, podemos citar o cadastro de Controle de Reservas, existente no Módulo de Faturamento.

Sintaxe:

Modelo2(cT,aC,aR,aGd,nOp,cLOk,cTOk, aGetsD, bF4, cIniCpos)

Parâmetros:

lRet = Retorno da função (.T. se confirmou);
cT = Título da Janela;
aC = Array com os Campos do Cabeçalho;
aR = Array com os Campos do Rodapé;
aGd = Array com as posições para edição dos ítens (GETDADOS);
nOp = Modo de operação (3 ou 4 altera e inclui ítens, 6 altera mas não inclui ítens, qualquer outro número só visualiza os ítens);
cLOk = Função para validação da linha;
cTOk = Função para validação de tudo (na confirmação);
aGetsD = Array com Gets editáveis;
bF4 = Bloco de Códigos, para tecla <F4>;
cIniCpos = String com Nome dos Campos que devem ser inicializados ao pressionar a tecla de Seta para Baixo.

Observação:

Lembre-se de utlizar o IDE, para criar, compilar e debugar o seu programa.
Tudo que esta entre /* e */ ou // é comentário.
Altere somente o que estiver em Negrito

Exercício 45- Criando a função Mod2Manut

```
User Function Mod2Manut(cAlias,nReg,nOpc)
//————————————————————+
//| Declaração de variáveis |
//————————————————————+
Local cChave := ""
Local nCOLS := 0
Local i := 0
Local lRet := .F.

//————————————————————————————————————————————————+
//| Variáveis que servirão de parâmetros para a função Modelo2() |
//————————————————————————————————————————————————+
Private cTitulo := cCadastro
Private aC := {}
Private aR := {}
Private aCGF := {}
Private cLinOk := ""
Private cAllOk := "u_Md2TudOk()"
Private aGetsGD := {}
Private bF4 := {|| }
Private cIniCpos := "+Z4_ITEM"
Private nMax := 99
Private aCordW := {}
Private lDelGetD := .T.
```

```
Private aHeader := {}
Private aCOLS := {}
Private nCount := 0
Private bCampo  := {|nField| FieldName(nField) }
Private dData := Ctod("  /  / ")
Private cNumero := Space(6)
Private aAlt := {}

//————————————————————————+
//| Cria variaveis de memoria M->??? |
//————————————————————————+

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

//————————————————————————————————————————+
//| Criar o vetor aHeader conforme o dicionario de dados |
//————————————————————————————————————————+

dbSelectArea("SX3")
dbSetOrder(1)
dbSeek(cAlias)
While !Eof() .And. SX3->X3_ARQUIVO == cAlias
If X3Uso(X3_USADO) .And. cNivel >= X3_NIVEL .And. !(Trim(X3_CAMPO)$"Z4_NUMERO/Z4_EMISSAO")
aAdd(aHeader,{ TRIM(X3_TITULO)  ,;
               X3_CAMPO      ,;
               X3_PICTURE    ,;
               X3_TAMANHO    ,;
               X3_DECIMAL    ,;
               X3_VALID      ,;
               X3_USADO      ,;
               X3_TIPO       ,;
               X3_ARQUIVO    ,;
               X3_CONTEXT    })
Endif
dbSkip()
End
dbSelectArea(cAlias)
dbSetOrder(1)

//————————————————————————————————————————————————————————————+
//| Se a opcao for diferente de incluir, entao atribuir os dados no vetor aCOLS |
//| Caso contrario criar o vetor aCOLS com a caracteristica de cada campo      |
//————————————————————————————————————————————————————————————+
If nOpc <> 3
cNumero := (cAlias)->Z4_NUMERO

  //————————————————————————————————————————————————————————————+
//| Deve-se posicionar porque nunca se sabe em qual item esta, e necessario pegar todos os itens |
  //————————————————————————————————————————————————————————————+
```

```
dbSeek(xFilial(cAlias)+cNumero)
While !Eof() .And. (cAlias)->(Z4_FILIAL+Z4_NUMERO) == Filial(cAlias)+cNumero

    //—————————————————————————————————————————————+
//| Criar o vetor com sua devida dimensão em relação ao dicionário de dados |
    //—————————————————————————————————————————————+

aAdd(aCOLS,Array(Len(aHeader)+1))
nCOLS++

    //———————————————————————————+
//| Atribuir o dado para cada coluna do vetor |
    //———————————-———————————————+

For i := 1 To Len(aHeader)
If aHeader[i,10]<>"V"
aCOLS[nCOLS,i] := FieldGet(FieldPos(aHeader[i,2]))
Else
aCOLS[nCOLS,i] := CriaVar(aHeader[i,2],.T.)
Endif
Next i

    //—————————————————————————————————————————————+
//| Criar uma última coluna para o controle da Getdados, se deletado ou não |
    //—————————————————————————————————————————————+
aCOLS[nCOLS,Len(aHeader)+1] := .F.
    //—————————————————————————————————————————————+
//| Atribuir o número do registro neste vetor para o controle na gravação |
    //—————————————————————————————————————————————+

aAdd(aAlt,RecNo())
dbSelectArea(cAlias)
dbSkip()
End
Else

  //———————————————————————————————————+
//| Atribuir à variável o inicializador padrão do campo |
  //———————————————————————————————————+
cNumero := GetSxeNum("SZ4",""Z4_NUMERO")
  //—————————————————————————————————————————————+
//| Criar o vetor com sua devida dimensão em relação ao dicionário de dados |
  //—————————————————————————————————————————————+
aAdd(aCOLS,Array(Len(aHeader)+1))
For i := 1 To Len(aHeader)
aCOLS[1,i] := CriaVar(aHeader[i,2])
Next i
  //—————————————————————————————————————————————+
```

```
//| Criar uma última coluna para o controle da Getdados, se deletado ou não |
  //————————————————————————————————————————————————————————————+
aCOLS[1,Len(aHeader)+1] := .F.

  //————————————————————————————————————+
//| Atribuir 01 para a primeira linha da Getdados |
  //————————————————————————————————————+

aCOLS[1,aScan(aHeader,{|x|Trim(x[2])=="Z4_ITEM"})] := "01"
Endif

//————————————————————————————————————————————+
//| Característica do vetor                    |
//| aC[n,1] = Nome da Variavel Ex.:"cCliente"   |
//| aC[n,2] = Array com coordenadas do Get [x,y] em PIXEL |
//| aC[n,3] = Titulo do Campo                  |
//| aC[n,4] = Picture                          |
//| aC[n,5] = Validacao                        |
//| aC[n,6] = F3                               |
//| aC[n,7] = Se campo e' editavel .t. se nao .f.  |
//————————————————————————————————————————————+

aAdd(aC,{"cNumero" ,{15,10},"Número",""@!",,,.F.})
aAdd(aC,{"dData",{15,80},"Data de Emissao",""99/99/99",,,(nOpc==3)})

//————————————————————————+
//| Coordenada do objeto Getdados |
//————————————————————————+
aCGD:={34,5,128,315}

//————————————————————————————————————————————————————————+
//| Validacao na mudanca de linha e quando clicar no botao Ok |
//————————————————————————————————————————————————————————+
cLinOk := "AllwaysTrue()"

//————————————————————————————————————+
//| Atribuir a database do sistema a variável |
//————————————————————————————————————+
dData := dDataBase

//————————————————————————+
//| Executar a função Modelo2() |
//————————————————————————+
lRet := Modelo2(cTitulo,aC,aR,aCGD,nOpc,cLinOk,cAllOk,,,cIniCpos,nMax)

//————————————+
//| Se confirmado |
//————————————+
If lRet
  //————————————————————+
//| Se opção for inclusao |
  //————————————————————+
```

```
If nOpc == 3
If MsgYesNo("Confirma gravação dos dados ?",cTitulo)
Processa({|| Md2Inclu(cAlias)},cTitulo,"Gravando os dados, aguarde...")
Endif

  //————————————————————+
//| Se opção for alteração |
//————————————————————+

Elseif nOpc == 4
If MsgYesNo("Confirma alteração dos dados ?",cTitulo)
Processa({|| Md2Alter(cAlias)},cTitulo,"Alterando os dados, aguarde...")
Endif

  //————————————————————+
//| Se opção for exclusão |
  //————————————————————+

Elseif nOpc == 5
If MsgYesNo("Confirma eliminação dos dados ?",cTitulo)
Processa({|| Md2Exclu(cAlias)},cTitulo,"Excluíndo os dados, aguarde...")
Endif
Endif
Else

  //————————————————————————————————————————————————+
//| Se não confirmado, reestabelecer a númeração automática do cadastro |
  //————————————————————————————————————————————————+

RollBackSX8()
Endif
Return
```

Exercício 46 – Criando a função Md2Inclu

```
Rotina de cadastro para a tabela SZ1.
Cadastro de software.
Função   Md2Inclu
Desc.    Função para incluir os dados

Static Function Md2Inclu(cAlias)
Local i := 0
Local y := 0

ProcRegua(Len(aCOLS))

dbSelectArea(cAlias)
dbSetOrder(1)
For i := 1 To Len(aCOLS)
IncProc()
If !aCOLS[i,Len(aHeader)+1]
RecLock(cAlias,.T.)
```

```
For y := 1 To Len(aHeader)
FieldPut(FieldPos(Trim(aHeader[y,2])),aCOLS[i,y])
Next y
(cAlias)->Z4_FILIAL  := xFilial(cAlias)
(cAlias)->Z4_NUMERO  := cNumero
(cAlias)->Z4_EMISSAO := dData
MsUnLock()
Endif
Next i
Return
```

Exercício 47 – Criando a função Md2Alter

```
Static Function Md2Alter(cAlias)
Local i := 0
Local y := 0

ProcRegua(Len(aCOLS))
dbSelectArea(cAlias)
dbSetOrder(1)
For i:=1 To Len(aCOLS)
If i<=Len(aAlt)
dbGoTo(aAlt[i])
RecLock(cAlias,.F.)
If aCOLS[i,Len(aHeader)+1]
dbDelete()
Else
For y := 1 To Len(aHeader)
FieldPut(FieldPos(Trim(aHeader[y,2])),aCOLS[i,y])
Next y
Endif
MsUnLock()
Else
If !aCOLS[i,Len(aHeader)+1]
RecLock(cAlias,.T.)
For y := 1 To Len(aHeader)
FieldPut(FieldPos(Trim(aHeader[y,2])),aCOLS[i,y])
Next y
(cAlias)->Z4_FILIAL  := xFilial(cAlias)
(cAlias)->Z4_NUMERO  := cNumero
(cAlias)->Z4_EMISSAO := dData
MsUnLock()
Endif
Endif
Next i
Return
```

Exercício 48– Criando a função Md2Exclu.

```
Static Function Md2Exclu(cAlias)

ProcRegua(Len(aCOLS))
dbSelectArea(cAlias)
dbSetOrder(1)
dbSeek(xFilial(cAlias)+cNumero)
While !Eof() .And. (cAlias)->Z4_FILIAL == xFilial(cAlias) .And. (cAlias)->Z4_NUMERO == cNumero
IncProc()
RecLock(cAlias,.F.)
dbDelete()
MsUnLock()
dbSkip()
End
Return
```

Exercicio 49- Criando a Função MD2TUDOK

```
User Function Md2TudOk()
Local lRet := .T.
Local i := 0
Local nDel := 0

For i:=1 To Len(aCOLS)
If aCOLS[i,Len(aHeader)+1]
nDel++
Endif
Next i
If nDel == Len(aCOLS)
MsgInfo("Para excluir todos os itens, utilize a opção EXCLUIR",cTitulo)
lRet := .F.
Endif
Return(lRet)
```

Observação:

Verifique que o "Tipo de Cadastro" criado é totalmente diferente dos utilizados até o momento.

4. Teste-o, verificando as sua "Validações" e saia do "Cadastro".

Anotações

Modelo3()

Esta Função é utilizada para a Criação de Telas de Cadastros, onde poderemos trabalhar com um esquema de Dois Arquivos em Tela, ou seja, um para a Criação do Cabeçalho e outro para a Criação do Corpo do Cadastro, onde iremos utilizar Linhas para a Digitação.

Como exemplo, podemos citar o cadastro de Pedidos de Vendas, existente no Módulo de Faturamento.

Sintaxe:

Modelo3(cTitulo,cAlias1,cAlias2,aMyEncho,cLinOk,cTudoOk,nOpcE,nOpcG,cFs2,aMyEncho, cLinOk,cTudoOk,nOpcE,nOpcG,cFieldOk,lVirtual,nLinhas,aAltEnchoice)

Parâmetros:

lRet	=	Retorno .T. Confirma / .F. Abandona;
cTitulo	=	Título da Janela;
cAlias1	=	Alias da Enchoice;
cAlias2	=	Alias da GetDados;
aMyEncho	=	Array com campos da Enchoice;
cLinOk	=	LinOk;
cTudOk	=	TudOk;
nOpcE	=	nOpc da Enchoice;
nOpcG	=	nOpc da GetDados;
cFieldOk	=	Validação para todos os campos da GetDados;
lVirtual	=	Permite visualizar campos virtuais na Enchoice;
nLinhas	=	Número Máximo de linhas na Getdados;
aAltEnchoice	=	Array com campos da Enchoice Alteráveis.

Exercício 50– Como criar telas de cadastros:

```
#Include "Protheus.Ch"
User Function Exerc09()

//————————————————————————————————————————————+
//| Declaração de variaveis                                  |
//| Declarar as variaveis aRotina e cCadastro como Private, porque a função mBrowse está esperando |
//————————————————————————————————————————————+
Private cCadastro := "Orçamento de venda"
Private cAlias1 := "SZ2"
Private cAlias2 := "SZ3
Private aRotina := {}

//————————————————————————————————————————————+
//| Montar o vetor aRotina, obrigatorio para a utilização da funçã mBrowse() |
//————————————————————————————————————————————+
aAdd( aRotina,{"Pesquisar" ,"AxPesqui"   , 0 , 1 })
aAdd( aRotina,{"Visualizar","u_Mod3Manut", 0 , 2 })
aAdd( aRotina,{"Incluir"   ,"u_Mod3Manut", 0 , 3 })
aAdd( aRotina,{"Alterar"   ,"u_Mod3Manut", 0 , 4 })
aAdd( aRotina,{"Excluir"   ,"u_Mod3Manut", 0 , 5 })
```

```
//————————————————————————————————————————————+
//| Selecionar a tabela pai, ordem e posicionar no primeiro registro da ordem |
//————————————————————————————————————————————+
dbSelectArea(cAlias1)
dbSetOrder(1)
dbGoTop()

//————————————————————————————————————+
//| Executar a função mBrowse para a tabela mencionada |
//————————————————————————————————————+
mBrowse(,,,,cAlias1)
Return
```

Exercício 51 – Como criar telas de cadastros:

```
User Function Mod3Manut(cAlias,nRecNo,nOpc)
//———————————————————+
//| Declaração de variaveis |
//———————————————————+
Local i := 0
Local cLinOk := "AllWaysTrue"
Local cTudoOk := "u_Md3TudOk"
Local nOpcE := nOpc
Local nOpcG := nOpc
Local cFieldOk := "AllWaysTrue"
Local lVirtual := .T.
Local nLinhas := 99
Local nFreeze := 0
Local lRet := .T.

Private aCOLS := {}
Private aHeader := {}
Private aCpoEnchoice := {}
Private aAltEnchoice := {}
Private aAlt := {}

  //————————————————————————————————+
//| Criar as variaveis M->Z2_??? da tabela PAI |
  //————————————————————————————————+
RegToMemory(cAlias1,(nOpc==3))

  //————————————————————————————————+
//| Criar as variaveis M->Z2_??? da tabela FILHO |
  //————————————————————————————————+
RegToMemory(cAlias2,(nOpc==3))

  //————————————————————————————————+
//| Criar o vetor aHeader, que eh o vetor que tem as caracteristicas para os campos da Getdados |
  //————————————————————————————————————————————————————————————+
CriaHeader()

  //————————————————————————————————————————————————————————————+
```

```
//| Criar o vetor aCOLS, que eh o vetor que tem os dados preenchidos pelo usuario, relacionado com o
vetor aHeader |
  //————————————————————————————————————+
criaCOLS(nOpc)

  //————————————————————+
//| Executar a função Modelo3() |
  //————————————————————+

lRet:=Modelo3(cCadastro,cAlias1,cAlias2,aCpoEnchoice,cLinOk,cTudoOk,nOpcE,;
nOpcG,cFieldOk,lVirtual,nLinhas,aAltEnchoice,nFreeze)

  //——————————+
//| Se confirmado |
  //——————————+
If lRet
    //————————————————+
//| Se opção for inclusao |
    //————————————————+

If nOpc == 3
If MsgYesNo("Confirma gravação dos dados ?",cCadastro)
Processa({||GrvDados()},cCadastro,"Gravando os dados, aguarde...")
Endif

    //————————————————+
//| Se opção for alteração |
    //————————————————+

Elseif nOpc == 4
If MsgYesNo("Confirma alteração dos dados ?",cCadastro)
Processa({||AltDados()},cCadastro,"Alterando os dados, aguarde...")
Endif

    //————————————————+
//| Se opção for exclusão |
    //————————————————+

Elseif nOpc == 5
If MsgYesNo("Confirma eliminação dos dados ?",cCadastro)
Processa({||ExcluiDados()},cCadastro,"Excluíndo os dados, aguarde...")
Endif
Endif
Else

    //+————————————————————————————————————————————————————+
//| Se não confirmado, reestabelecer a númeração automática do cadastro |
    //+————————————————————————————————————————————————————+

RollBackSX8()
Endif
Return
```

Exercício 52 – Criando a função CriaHeader

```
Static Function CriaHeader()
aHeader:={}
aCpoEnchoice := {}
aAltEnchoice := {}

dbSelectArea("SX3")
dbSetOrder(1)
dbSeek(cAlias2)

While !Eof() .And. X3_ARQUIVO == cAlias2
If X3USO(X3_USADO) .And. cNivel >= X3_NIVEL
aAdd(aHeader,{TRIM(X3_TITULO),;
                X3_CAMPO,;
                X3_PICTURE,;
                X3_TAMANHO,;
                X3_DECIMAL,;
                X3_VALID,;
                X3_USADO,;
                X3_TIPO,;
                X3_ARQUIVO,;
                X3_CONTEXT})
Endif
dbSkip()
End

dbSeek(cAlias1)
While !Eof() .And. X3_ARQUIVO == cAlias1
If X3USO(X3_USADO) .And. cNivel >= X3_NIVEL
aAdd(aCpoEnchoice,X3_CAMPO)
aAdd(aAltEnchoice,X3_CAMPO)
Endif
dbSkip()
End
Return
```

Exercício 53 – Criando a função CriaCols

```
Static Function CriaCOLS(nOpc)
Local nQtdCpo := 0
Local i := 0
Local nCOLS := 0

nQtdCpo := Len(aHeader)
aCOLS := {}
aAlt := {}

If nOpc == 3
aAdd(aCOLS,Array(nQtdCpo+1))
For i := 1 To nQtdCpo
aCOLS[1,i] := CriaVar(aHeader[i,2])
```

```
Next i
aCOLS[1,nQtdCpo+1] := .F.
Else
dbSelectArea(cAlias2)
dbSetOrder(1)
dbSeek(xFilial(cAlias2)+(cAlias1)->Z2_NUMERO)
While !Eof() .And. (cAlias2)->Z3_FILIAL == xFilial(cAlias2) .And. (cAlias2)->Z3_NUMERO == (cAlias1)->Z2_
NUMERO
aAdd(aCOLS,Array(nQtdCpo+1))
nCOLS ++
For i := 1 To nQtdCpo
If aHeader[i,10] <> "V"
aCOLS[nCOLS,i] := FieldGet(FieldPos(aHeader[i,2]))
Else
aCOLS[nCOLS,i] := CriaVar(aHeader[i,2],.T.)
Endif
Next i
aCOLS[nCOLS,nQtdCpo+1] := .F.
aAdd(aAlt,RecNo())
dbSelectArea(cAlias2)
dbSkip()
End
Endif
Return
```

Exercício 54 – Criando a função GrvDados

Rotina de cadastro para a tabela SZ1.

Cadastro de software.

Função GrvDados
Desc. Função para incluir os dados na tabela FILHO conforme o vetor aHeader e aCOLS e também incluir os dados na tabelaPAI conforme as variáveis M->???

```
Static Function GrvDados()
Local bCampo := { |nField| Field(nField) }
Local i := 0
Local y := 0
Local nItem := 0
```

Anotações

```
ProcRegua(Len(aCOLS)+FCount())
dbSelectArea(cAlias1)
RecLock(cAlias1,.T.)
For i := 1 To FCount()
IncProc()
If "FILIAL" $ FieldName(i)
FieldPut(i,xFilial(cAlias1))
Else
FieldPut(i,M->&(Eval(bCampo,i)))
Endif
Next i
MsUnLock()

dbSelectArea(cAlias2)
dbSetOrder(1)
For i := 1 To Len(aCOLS)
IncProc()
If !aCOLS[i,Len(aHeader)+1]
RecLock(cAlias2,.T.)
For y := 1 To Len(aHeader)
FieldPut(FieldPos(Trim(aHeader[y,2])),aCOLS[i,y])
Next y
nItem++
(cAlias2)->Z3_FILIAL := xFilial(cAlias2)
(cAlias2)->Z3_NUMERO := (cAlias1)->Z2_NUMERO
(cAlias2)->Z3_ITEM   := StrZero(nItem,2,0)
MsUnLock()
Endif
Next i
Return
```

Exercício 55 – Criando a função AltDados

Rotina de cadastro para a tabela SZ1.
Cadastro de software.
Função AltDados
Desc. Função para incluir/alterar os dados na tabela FILHO conforme o vetor aHeader e aCOLS e também incluir os dados na tabela PAI conforme as variáveis M->???

```
Static Function AltDados()
Local bCampo := { |nField| Field(nField) }
Local i := 0
Local y := 0
Local nItem := 0

ProcRegua(Len(aCOLS)+FCount())
dbSelectArea(cAlias1)
RecLock(cAlias1,.F.)
For i := 1 To FCount()
IncProc()
If "FILIAL" $ FieldName(i)
FieldPut(i,xFilial(cAlias1))
```

```
Else
FieldPut(i,M->&(Eval(bCampo,i)))
Endif
Next i
MsUnLock()
dbSelectArea(cAlias2)
dbSetOrder(1)
nItem := Len(aAlt)+1
For i:=1 To Len(aCOLS)
If i<=Len(aAlt)
dbGoTo(aAlt[i])
RecLock(cAlias2,.F.)
If aCOLS[i,Len(aHeader)+1]
dbDelete()
Else
For y := 1 To Len(aHeader)
FieldPut(FieldPos(Trim(aHeader[y,2])),aCOLS[i,y])
Next y
Endif
MsUnLock()
Else
If !aCOLS[i,Len(aHeader)+1]
RecLock(cAlias2,.T.)
For y := 1 To Len(aHeader)
FieldPut(FieldPos(Trim(aHeader[y,2])),aCOLS[i,y])
Next y
(cAlias2)->Z3_FILIAL := xFilial(cAlias2)
(cAlias2)->Z3_NUMERO := (cAlias1)->Z2_NUMERO
(cAlias2)->Z3_ITEM   := StrZero(nItem,2,0)
MsUnLock()
nItem++
Endif
Endif
Next i
Return
```

Exercício 56– Criando a função ExcluiDados

```
Static Function ExcluiDados()

ProcRegua(Len(aCOLS)+1)

dbSelectArea(cAlias2)
dbSetOrder(1)
dbSeek(xFilial(cAlias2)+(cAlias1)->Z2_NUMERO)
While !Eof() .And. (cAlias2)->Z3_FILIAL == xFilial(cAlias2) .And. (cAlias2)->Z3_NUMERO == (cAlias1)->Z2_
NUMERO
IncProc()
RecLock(cAlias2,.F.)
dbDelete()
MsUnLock()
dbSkip()
```

```
End
dbSelectArea(cAlias1)
dbSetOrder(1)
IncProc()
RecLock(cAlias1,.F.)
dbDelete()
MsUnLock()
Return
```

Exercício 57 – Criando a função Md3TudOk

```
Rotina de cadastro para a tabela SZ1.
Cadastro de software.
Função  Md3TudOk
Desc.   Função para validar se a getdados está liberada para gravar os dados conforme o preenchimento

User Function Md3TudOk()
Local lRet := .T.
Local i := 0
Local nDel := 0

For i:=1 To Len(aCOLS)
If aCOLS[i,Len(aHeader)+1]
nDel++
Endif
Next i
If nDel == Len(aCOLS)
MsgInfo("Para excluir todos os itens, utilize a opção EXCLUIR",cCadastro)
lRet := .F.
Endif
Return(lRet)
```

O que você aprendeu neste capítulo

Neste capítulo aprendemos a utilizar as Funções de Criação de Telas para Cadastros, juntamente com Rotinas de Validações, utilizadas frequentemente durante a Elaboração de Programas.

Próximo Passo

Na próxima etapa, aprenderemos como criar Consultas Padrões do sistema, ou seja, as consultas acionadas através da tecla <F3> nos Campos.

Anotações

CRIAÇÃO DE RELATÓRIOS CONSULTAS PADRÃO

O que você irá aprender neste capítulo

Neste capítulo, aprenderemos a desenvolver os relatórios utilizando o IDE.

Rotinas Abordadas

• Geração de Relatórios.

O que você irá aprender neste capítulo

Neste capítulo, aprenderemos a criar Relatórios Customizados.

Rotinas abordadas

• Funções, para Relatórios:

– SetPrint();
– SetDefault();
– Pergunte();
– SetRegua();
– RptStatus();
– IncRegua().

• Criação de Relatórios Customizados;

• Criação de Relatórios com Relacionamento;

• Configuração do Arquivo de Perguntas e Respostas.

Variáveis de Relatórios

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

Variável	Tipo	Conteúdo
Wnrel	Local	Nome default do relatório em disco
CbCont	Local	Contador
Cabec1	Local	1ª linha do cabeçalho do relatório
Cabec2	Local	2ª linha do cabeçalho do relatório
Cabec3	Local	3ª linha do cabeçalho do relatório
Tamanho	Local	Tamanho do Relatório (P = Pequeno 80 Colunas, M= Médio 132 colunas),G = (Grande 220 colunas)
Limite	Local	Quantidade de colunas no relatório (80, 132, 220).
Titulo	Local	Titulo do Relatório
CDesc1	Local	1ª linha da descrição do relatório
CDesc2	Local	2ª linha da descrição do relatório
CDesc3	Local	3ª linha da descrição do relatório

aReturn	Private	Array com as informações para a tela de configuração da impressão
Nomeprog	Private	Nome do programa do relatório
Cstring	Private	Alias do arquivo principal do relatório para o uso de filtro
Li	Private	Controle das linhas de imp.Seu valor inicial é a qtda máxima de linhas por página utilizada no relatório.
M_Pag	Private	Controle do número de páginas do relatório.
Aord	Private	Array contendo as ordens de layout para a impressão.

Caso não existam várias ordens este array deve estar vazio. Ex.: aOrd:= {Código, Descrição, Telefone} -> O layout do relatório vai depender da ordem selecionada na tela de configuração de impressão.

nLastKey	Private	Utilizado para controlar o cancelamento da impressão do relatório.
Cperg	Private	Nome da pergunte a ser exibida para o usuário.
Alinha	Private	Array que contem informações para a impressão de relatórios cadastrais

Anotações

SetPrint()

Essa Função é a responsável pela criação da tela do dispositivo de impressão dos relatórios, montando a interface com o usuário.

Sintaxe:

SetPrint(ExpC2, ExpC3, ExpC4, ExpC5, ExpC6, ExpC7, ExpC8, ExpL1, ExpA1, ExpL2, ExpC9).

Parâmetros:

ExpC2	=	Alias do Arquivo Principal (Se existir);
ExpC3	=	Nome Padrão do Relatório;
ExpC4	=	Nome do Grupo de Perguntas;
ExpC5 .. ExpC8	=	Descrição do Relatório;
ExpL1	=	Habilita o Dicionário de Dados: .T. => Habilita (Só utilizar em conjunto com a Função ImpCadast); .F. => Desabilita.
ExpA1	=	Array contendo as Ordens de Indexação do Arquivo Principal;
ExpL2	=	Habilita a Alteração da Compressão do Relatório; .T. => Habilita; .F. => Desabilita.
ExpC9	=	Classificação do Relatório por Tamanho ("G","M" ou "P"): P = 80 colunas; M = 132 colunas; G = 220 colunas.

SetDefault()

Esta Função permite Habilitar os Padrões definidos pela Função Set Print().

Sintaxe:

SetDefault(Array,Alias).

Parâmetros:

Array	=	Array aReturn, preenchido pelo SetPrint;
Array1	=	Reservado para o Formulário;
Array2	=	Reservado para Nº de Vias;
Array3	=	Destinatário;
Array4	=	Formato (1–Comprimido, 2–Normal);
Array5	=	Mídia a ser Descarregada (1–Comprimido, 2–Normal);
Array6	=	Porta da Printer ou Arquivo (1–LPT1, 4–Com1);
Array7	=	Expressão ou Filtro de Dados;
Array8	=	Ordem a ser Selecionada;
Array9	=	1º Campo a Processar;
Array10	=	2º Campo a Processar;
Array11	=	3º Campo a Processar;
ArrayN	=	4º Campo a Processar;
Alias	=	Alias do Arquivo a ser Impresso.

Pergunte()

Esta Função permite acessar e editar um Grupo de Perguntas de Programas, baseando-se no Arquivo de Perguntas e Respostas do Sistema (Sx1), conhecido também como Parâmetros de Relatórios.

Mostra uma tela contendo um Lote de Parâmetros a serem respondidos e em seguida confirmados pelo usuário.

Sintaxe:

Pergunte(cPergunta,lAsk).

Parâmetros:

cPergunta	=	Código da Pergunta no SX1 conforme o Programa / Nome do Grupo de Perguntas;
lAsk	=	Mostrar a Tela de Perguntas ou simplesmente carrega os Valores Default. .F. = Devolve Conteúdo da Variáveis, não apresentando a Janela de Perguntas; .T. = Permite Alteração das Variáveis apresentando a Janela.

Anotações

SetRegua()

Inicializa a Régua Padrão, para acompanhar a Evolução do Processamento dos Relatórios.

Sintaxe:

SetRegua(n1).

Parâmetros:

<n1> = Numérico, Número de registros que serão processados.

Anotações

RptStatus()

Executa a Função de Detalhes do Relatório, ou seja, é a Impressão da Página propriamente dita.

Sintaxe:

RptStatus(b1).

Parâmetros:

<b1> = Bloco de Código que define a Função a ser executada.
Comentários:
Pode ser utilizada com o Parâmetro:
RptStatus({|| ("Nome da Função") })

IncRegua()

Incrementa a Régua Padrão de Processamento em Relatórios, ou seja, é a responsável por sincronizar a Barra de Progressão de acordo com o número de registros selecionados para a impressão.

Sintaxe:

IncRegua().

Parâmetros:

Nil.

Relatórios

A Criação de Relatórios no Protheus é muito facilitado pelo Assistente de Código, porém sempre haverá a necessidade de Complementos Adicionais, pois a ferrramenta cria apenas o Fonte Padrão.

Exercício 58– Como Criar Relatórios, para Cadastro de Softwares, através do Assistente de Código da Ferramenta – IDE:

1. Acesse a "Ferramenta – IDE";

2. Selecione as seguintes opções "Ferramentas" + "Assistente de Código" + "Relatório";

3. Clique na opção "Avançar>>";

4. Selecione a opção "Arquivo Padrão";

5. No "Campo – Alias", informe "SZ1";

6. Selecione a opção "Utiliza Índice Padrão";

7. No "Campo – Ordem", informe "1" e clique na opção "Avançar>>";

8. No "Campo – Chave", informe "RSZ1";

9. Clique na opção "Avançar>>";

10. No "Campo – Título", informe "Relatório de Softwares";

11. No "Campo – Cabec1", informe "Código Nome;

12. No "Campo – Cabec2", informe "------ ----------";

13. Selecione a opção "Normal";

14. No "Campo – Tamanho", selecione "80 Colunas";

15. Na pasta "Ordem", posicione com o cursor sobre o "Campo – Ordem", informe "Código" e clique na opção "adicionar";

16. Na sequência, informe também "Nome" e clique na opção "adicionar";

17. Nas pastas "Habilitações", deixe tudo selecionado e clique na opção "Finalizar".

Anotações

Exercício 59 – Como realizar Alterações necessárias nos Fontes:

1. Realize as "Alterações" necessárias, seguindo o exemplo a seguir, de acordo com o "Texto em Negrito":

```
#Include "rwmake.ch"
*********************************************************************************
*  Programa  |  RPAD  |  Autor | Ferramenta IDE    | Data |  99/99/99
*********************************************************************************
*  Descrição | Código gerado pelo Protheus 8 IDE.
*********************************************************************************
User Function Exerc18
Private cString
Local aOrd              :=      {}
Private CbTxt           :=      ""
Local cDesc1            :=      "Este programa tem como objetivo imprimir relatórios"
Local cDesc2            :=      "de acordo com os parâmetros informados pelo usuário."
Local cDesc3            :=      "Cadastro de Software"
Local cPict             :=      ""
Private lEnd            :=      .F.
Private lAbortPrint     :=      .F.
Private limite          :=      80
Private tamanho         :=      "P"
Private nomeprog        :=      "Exerc18" // Coloque aqui o Nome do Programa para impressão
no cabeçalho
Private nTipo           :=      18
Private aReturn         :=      {"Zebrado", 1, "Administração", 2, 2, 1, "", 1}
Private nLastKey        :=      0
Private cPerg           :=      "RSZ1"  // Nomo do Grupo de Perguntas
Local titulo            :=      "Cadastro de Software"
Local nLin              :=      80
Local Cabec1            :=      "Codigo Nome"
Local Cabec2            :=      "———————— ———————"
Private cbtxt           :=      Space(10)
Private cbcont          :=      00
Private CONTFL          :=      01
Private m_pag           :=      01
Local imprime           :=      .T.
Private wnrel           :=      "Exerc18" // Coloque aqui o Nome do Arquivo usado para
impressão em disco.
Private cString         :=      "SZ1"
dbSelectArea("SZ1")
dbSetOrder(1)
pergunte(cPerg,.F.)
*********************************************************************************
*  Monta a interface padrão com o usuário...
*********************************************************************************
wnrel  :=  SetPrint(cString,NomeProg,cPerg,@titulo,cDesc1,cDesc2,cDesc3,.T.,aOrd,.T.,Tamanho,,.
T.)
If nLastKey == 27
        Return
Endif
SetDefault(aReturn,cString)
```

```
If nLastKey == 27
  Return
Endif
nTipo := If(aReturn[4]==1,15,18)
*****************************************************************************
*  Processamento. RPTSTATUS monta janela com a régua de processamento.
*****************************************************************************
RptStatus({|| RunReport(Cabec1,Cabec2,Titulo,nLin) },Titulo)
RETURN
*****************************************************************************
*  Função   | RUNREPORT | Autor | Protheusx IDE          | Data | 99/99/99
*****************************************************************************
*  Descrição | Função auxiliar chamada pela RPTSTATUS. A função RPTSTATUS º
*               | monta a janela de processamento
*****************************************************************************
Static Function RunReport(Cabec1,Cabec2,Titulo,nLin)
Local nOrdem
dbSelectArea(cString)
dbSetOrder(1)
*****************************************************************************
| * SETREGUA -> Indica quantos registros serão processados para a régua
*****************************************************************************
SetRegua(RecCount())
*****************************************************************************
dbGoTop()
DBSEEK(xFilial("SZ1")+mv_par01,.T.)
While !EOF() .and. SZ1->Z1_Codigo <= MV_Par02
*****************************************************************************
* Verifica o cancelamento pelo usuário...
*****************************************************************************
 If lAbortPrint
   @nLin,00 PSAY "*** Cancelado pelo Operador ***"
   Exit
  Endif
*****************************************************************************
* Impressão do cabeçalho do relatório. . .
*****************************************************************************
 If nLin > 55 // Salto de Página. Neste caso o formulário tem 55 linhas...
    Cabec(Titulo,Cabec1,Cabec2,NomeProg,Tamanho,nTipo)
    nLin := 8
  Endif
//* Coloque aqui a lógica da impressão do seu programa.Utilize PSAY para
saída na impressora. Por exemplo:*
nLin := nLin + 1 // Avança a linha de impressão
@ nLin,000 PSay SZ1->Z1_CODIGO
@ nLin,008 PSay SZ1->Z1_NOME
@ nLin,061 PSay SZ1->Z1_EMISSAO
@ nLin,071 PSay SZ1->Z1_DTAQUIS

dbSkip() // Avança o ponteiro do registro no arquivo
Incregua()
EndDo
```

```
*****************************************************************************
* Finaliza a execucao do relatório...
*****************************************************************************
SET DEVICE TO SCREEN  // Retorna para a Tela
If aReturn[5]==1    // Se impressão em disco, chama o gerenciador de impressão...
  dbCommitAll()
 SET PRINTER TO
 OurSpool(wnrel)
Endif
MS_FLUSH() // Limpa o Buffer de Impressão
Return
```

2. Confira os dados, salve o "Programa"

Anotações

Perguntas e Respostas

Todos os Relatórios existentes no Protheus necessitam de Perguntas, para imprimir corretamente os dados, ou seja, é necessário Configurarmos um Arquivo, para parametrizar o que desejamos imprimir.

O Arquivo responsável por essa Configuração é o (Sx1) e ele servirá tanto para as Perguntas e Respostas de Relatórios, quanto para as Rotinas de Processamentos.

Dica

Lembre-se que o mesmo deverá criar as Perguntas pelo "Módulo Configurador".

Para anular a "Impressão dos Parâmetros dos Relatórios", acesse o "Cadastro de Parâmetros", no "Módulo Configurador" e configure os "Parâmetros – MV_IMPSX1, MV_SALTPAG e MV_CANSALT", com o "Conteúdo" igual a "N", dessa maneira as "Perguntas", não serão mais impressas e também não haverá "Saltos" e tampouco emissão de "Páginas em Branco".

Exercício 60 – Desenvolvendo o relatório da tabela SZ1 e SZ2, usando a função "POSICIONE", e criando as perguntas pela função CRIASX1

Dica

Lembre-se que o mesmo deverá utilizar o IDE, para desenvolver o código.

```
User Function Exerc11()
//+-------------------------
//| Declaracoes de variaveis
//+-------------------------
```

```
Local cDesc1        := "Este relatorio ira imprimir informacoes do orçamento de software conforme"
Local cDesc2        := "parâmetros informados pelo usuário"
Local cDesc3        := "[Utilizando Posicionamento e Condições p/ Registros]"

Private cString     := "SZ2"
Private Tamanho     := "M"
Private aReturn     := { "Zebrado",1,"Administracao",2,2,1,"",1 }
Private wnrel       := "EXERC11"
Private NomeProg    := wnrel
Private nLastKey    := 0
Private Limite      := 132
Private Titulo      := "Orcamento de software"
Private cPerg       := "EXEC11"
Private nTipo       := 0
Private cbCont      := 0
Private cbTxt       := "registro(s) lido(s)"
Private Li          := 80
Private m_pag       := 1
Private aOrd        := {}
Private Cabec1      := "Código Item Descricao do Software  Quantidade Vlr.Unitario Vlr.Total"
Private Cabec2      :=       ""

CriaSx1()

//————————————————————————————-
//| Disponibiliza para usuario digitar os parametros
//————————————————————————————-
Pergunte(cPerg,.F.)

//————————————————————————————————
//| Solicita ao usuario a parametrizacao do relatorio
//————————————————————————————————
wnrel:=SetPrint(cString,wnrel,cPerg,@Titulo,cDesc1,cDesc2,cDesc3,.F.,aOrd,.F.,;
Tamanho,.F.,.F.)
//————————————————————————————————————————————————————————————-
//| Parâmetros da função SetPrint
//| SetPrint(cAlias,cNome,cPerg,cDesc,cCnt1,cCnt2,cCnt3,lDic,aOrd,lCompres,;
//| cSize,aFilter,lFiltro,lCrystal,cNameDrv,lNoAsk,lServer,cPortToPrint)
//————————————————————————————————————————————————————————————-

//————————————
//| Se teclar ESC, sair
//————————————
If nLastKey == 27
Return
Endif

//+————————————————————————————————————————————————————
//| Estabelece os padroes para impressao, conforme escolha do usuario
//+————————————————————————————————————————————————————
SetDefault(aReturn,cString)
```

```
//+-------------------------------------------------
//| Verificar se sera reduzido ou normal
//+-------------------------------------------------
nTipo := Iif(aReturn[4] == 1, 15, 18)

//+-----------------
//| Se teclar ESC, sair
//+-----------------
If nLastKey == 27
Return
Endif

//+-------------------------------------------------
//| Chama funcao que processa os dados
//+-------------------------------------------------
RptStatus({|lEnd| Exerc11Imp(@lEnd) },"Aguarde...","Imprimindo os dados...", .T.)

Return

Static Function Exerc11Imp(lEnd)
Local nTPedido  := 0

//-----------------------------------
//| Selecionar e posicionar na tabela principal
//-----------------------------------

dbSelectArea("SZ2")
dbSetOrder(1)
dbSeek(xFilial("SZ2")+mv_par01,.T.)

While !Eof() .And. SZ2->(Z2_FILIAL+Z2_NUMERO) <= xFilial("SZ2")+mv_par02
If lEnd
@ Li,000 PSay cCancel
Exit
Endif

//-----------------------------------------------------------------
//| Se o registro estiver fora dos parâmetros, ir para o próximo registro
//-----------------------------------------------------------------

If SZ2->Z2_CLIENTE < mv_par03 .Or. SZ2->Z2_CLIENTE > mv_par04
```

Anotações

```
dbSkip()
Loop
Endif

//————————————————————————————————————————————
//| Se o registro estiver fora dos parâmetros, ir para o próximo registro
//————————————————————————————————————————————

If SZ2->Z2_EMISSAO < mv_par05 .Or. SZ2->Z2_EMISSAO > mv_par06
dbSkip()
Loop
Endif

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

@ Li,001 PSay "No. Orcamento:"+SZ2->Z2_NUMERO
@ Li,024 PSay "Cliente:"+SZ2->(Z2_CLIENTE+""+Z2_LOJA)+""+Posicione("SA1",1,xFilial("SA1")+SZ2->(Z2_
CLIENTE+Z2_LOJA),"A1_NOME")
@ Li,093 PSay "Emissao:"+Dtoc(SZ2->Z2_EMISSAO)
@ Li,113 PSay "Validade:"+Dtoc(SZ2->Z2_DTVALID)
Li+=2

dbSelectArea("SZ3")
dbSetOrder(1)
dbSeek(xFilial("SZ3")+SZ2->Z2_NUMERO)
While !Eof() .And. SZ3->Z3_FILIAL+SZ3->Z3_NUMERO == xFilial("SZ3")+SZ2->Z2_NUMERO
If lEnd
@ Li,000 PSay cCancel
Exit
Endif

If Li > 55

Cabec(Titulo,Cabec1,Cabec2,NomeProg,Tamanho,nTipo)
Endif

@ Li,001 PSay SZ3->Z3_CODSOFT
@ Li,013 PSay SZ3->Z3_ITEM
@ Li,021 PSay Posicione("SZ1",1,xFilial("SZ1")+SZ3->Z3_CODSOFT,"Z1_NOME")
@ Li,077 PSay SZ3->Z3_QUANT  PICTURE "@E 999,999,999.99"
@ Li,097 PSay SZ3->Z3_UNIT   PICTURE "@E 999,999,999.99"
@ Li,117 PSay SZ3->Z3_VLRTOT PICTURE "@E 999,999,999.99"
nTPedido += SZ3->Z3_VLRTOT
Li++
dbSelectArea("SZ3")
dbSkip()
End
@ Li,000 PSay __PrtThinLine()
```

```
Li++
@ Li,000 PSay "Total do Pedido:"
@ Li,117 PSay nTPedido PICTURE "@E 999,999,999.99"
nTPedido := 0
Li++
@ Li,000 PSay __PrtThinLine()
Li+=2

dbSelectArea("SZ2")
dbSkip()
End
dbSelectArea("SZ2")
dbSetOrder(1)
dbGoTop()

dbSelectArea("SZ3")
dbSetOrder(1)
dbGoTop()

If aReturn[5] == 1
Set Printer TO
dbCommitAll()
Ourspool(wnrel)
EndIf
Ms_Flush()
Return
Static Function CriaSx1()
Local aHelp := {}
aAdd( aHelp ,{{"Informe o número do orçcamento inicial"} ,{""} ,{""}} )
aAdd( aHelp ,{{"Informe o número do orçamento final"}   ,{""} ,{""}} )
aAdd( aHelp ,{{"Informe o código do cliente de início"} ,    {""} ,{""}} )
aAdd( aHelp ,{{"Informe o código do cliente para fim"}  ,    {""} ,{""}} )
aAdd( aHelp ,{{"Emitir os orçamento a partir da data"}  ,    {""} ,{""}} )
aAdd( aHelp ,{{"Emitir os orçamento até a data"}       ,     {""} ,{""}} )
//texto do help português                                  inglês      espanhol

PutSx1(cPerg,"01","Orcamento de?","","","mv_ch1","C",06,00,00,";
G","","","","","mv_par01","","","","","","","","","","","","","","","",";
aHelp[1,1],aHelp[1,2],aHelp[1,3],"")

PutSx1(cPerg,"02","Orcamento ate?","","","","mv_ch2","C",06,00,00,;
"G","","","","","mv_par02","","","","","","","","","","","","","","","",";
aHelp[2,1],aHelp[2,2],aHelp[2,3],"")

PutSx1(cPerg,"03","Cliente de?","","","","mv_ch3","C",06,00,00,"G","","","SA1",;
"","","","mv_par03","","","","","","","","","","","","","","","",";
aHelp[3,1],aHelp[3,2],aHelp[3,3],"")

PutSx1(cPerg,"04","Clienteate?","","","","mv_ch4","C",06,00,00,"G","","","SA1",;
"","","","mv_par04","","","","","","","","","","","","","","","",aHelp[4,1],;
aHelp[4,2],aHelp[4,3],"")
```

```
PutSx1(cPerg,"05","Emissao de?","","","mv_ch5","D",08,00,00,"G","","","","","",;
"mv_par05","","","","","","","","","","","","","","","",aHelp[5,1],aHelp[5,2],;
aHelp[5,3],"")

PutSx1(cPerg,"06","Emissao ate?","","","mv_ch6","D",08,00,00,"G","","","","","",;
"mv_par06","","","","","","","","","","","","","","","",aHelp[6,1],aHelp[6,2],;
aHelp[6,3],"")
Return
```

O que você aprendeu neste capítulo

Neste capítulo, você aprendeu a criar Relatórios Personalizados com a utilização de um ou mais Arquivos Relacionados, através de Funções da própria linguagem.

Próximo Passo

No próximo capítulo,iremos abordar o processo de gerar um código para importação de dados.Importação de Dados

Anotações

Sintaxe protheus x sql

```
Activate Dialog oLeTxt Centered
Return
//Função OKLETXT  Autor  AP8 IDE          Data   10/02/05
//Descrição Funcao chamada pelo botao OK na tela inicial de processamento
//Executa a leitura do arquivo texto.
//Uso Programa principal

Static Function OkLeTxt
// Abertura do arquivo texto
Private cArqTxt := "D:\AP8\SYSTEM\CLIENTES.TXT" //Verificar onde o arquivo esta gravado
Private nHdl    := fOpen(cArqTxt,68)
Private cEOL    := "CHR(13)+CHR(10)"
If Empty(cEOL)
   cEOL := CHR(13)+CHR(10)
Else
   cEOL := Trim(cEOL)
   cEOL := &cEOL
Endif

If nHdl == -1
   MsgAlert("O arquivo de nome "+cArqTxt+" nao pode ser aberto! Verifique os parametros.","Atencao!")
   Return
Endif
```

```
//Inicializa a regua de processamento
Processa({|| RunCont() },"Processando...")
Return
//Função   RUNCONT  Autor  AP8 IDE          Data   10/02/05
//Descrião Funcao auxiliar chamada pela PROCESSA.  A funcao PROCESSA
//monta a janela com a regua de processamento.
//Uso  Programa principal

Static Function RunCont

Local nTamFile, nTamLin, cBuffer, nBtLidos
// Lay-Out do arquivo Texto gerado:

//Campo         Inicio              Tamanho
// A6_FILIAL     01                  02
// A6_COD       02                  06
// A6_LOJA       03                  02
// A6_NOME     04                  40
// A6_PESSOA   05                  1
// A6_NREDUZ 06                  20
// A6_END        07                   40
// A6_TIPO       08                   1
// A6_MUN      09                  15

nTamFile := fSeek(nHdl,0,2)
fSeek(nHdl,0,0)
nTamLin  := 127+Len(cEOL)
cBuffer  := Space(nTamLin) // Variavel para criacao da linha do registro para leitura
nBtLidos := fRead(nHdl,@cBuffer,nTamLin) // Leitura da primeira linha do arquivo texto
ProcRegua(nTamFile) // Numero de registros a processar
While nBtLidos >= nTamLin

   // Incrementa a regua
      IncProc()
    // Grava os campos obtendo os valores da linha lida do arquivo texto.

dbSelectArea(cString)
RecLock(cString,.T.)

   SA1->A1_FILIAL := Substr(cBuffer,01,02) //Identificar a posição do arquivo TXT
   SA1->A1_COD := Substr(cBuffer,05,07) //Identificar a posição do arquivo TXT
   SA1->A1_LOJA := Substr(cBuffer,11,15)//Identificar a posição do arquivo TXT
   SA1->A1_NOME := Substr(cBuffer,22,29) //Identificar a posição do arquivo TXT
   SA1->A1_NREDUZ := Substr(cBuffer,33,44) //Identificar a posição do arquivo TXT
   SA1->A1_END:=Substr(cBuffer,33,44) //Identificar a posição do arquivo TXT
   SA1->A1_TIPO:=Substr(cBuffer,33,44) //Identificar a posição do arquivo TXT
   SA1->A1_MUN:=Substr(cBuffer,33,44) //Identificar a posição do arquivo TXT
   SA1->A1_EST:=Substr(cBuffer,33,44) //Identificar a posição do arquivo TXT
MSUnLock()
      //Leitura da proxima linha do arquivo texto.
   nBtLidos := fRead(nHdl,@cBuffer,nTamLin) // Leitura da proxima linha do arquivo texto
   dbSkip()
```

```
EndDo
//O arquivo texto deve ser fechado, bem como o dialogo criado na função anterior
fClose(nHdl)
//Close(oLeTxt) // Gerado pelo IDE
Return
```

O que você aprendeu neste capítulo

Neste capítulo, você aprendeu a criar Relatórios Personalizados com a utilização de um ou mais Arquivos Relacionados, através de Funções da própria linguagem.

Próximo Passo

Neste capítulo aprenderemos como deverão ser criados os Pontos de Entradas e a implementação dos mesmos, junto as rotinas padrões do sistema.

Anotações

SINTAXE PROTHEUS X SQL

O que você irá aprender neste capítulo

Neste capítulo, iremos abordar a sintaxe Protheus X SQL.

Rotinas abordadas

Como padrão de escrita para as consultas SQL e para as Stored Procedures desenvolvidas na Microsiga, adotou-se o TRANSACT-SQL (T-SQL) com pequenas alterações para poder converter a sintaxe para os demais bancos.

Esta conversão é feita através de uma função desenvolvida pela Microsiga e que visa simplificar a escrita de comandos SQL, pois o desenvolvedor não precisará saber a sintaxe de todos os bancos com que o Protheus trabalha, bastando conhecer uma linguagem e as pequenas alterações que nela foram feitas

Particularidades do Protheus

Quando utilizamos SGBD's no Protheus, existem algumas particularidades que devem ser observadas:

Coluna R_E_C_N_O_

Utilizado como chave primária dentro das tabelas da base Protheus. Criado para manter a compatibilidade com a biblioteca de acesso anterior (DBF), permitindo a utilização das funções de posicionamento de linhas (Recno()) em ambiente SQL.

Coluna D_E_L_E_T_

A exclusão de registros na base Protheus ocorre de maneira lógica, ou seja, quando a aplicação executa um comando de exclusão, o banco recebe um comando de atualização do conteúdo da coluna D_E_L_E_T_, passando esse registro para o estado "excluído".
Nesse momento, o registro mesmo excluído ainda está na base de dados. Sua eliminação física depende de um comando PACK (ADVPL), ou DELETE (SQL).

Anotações

Exercício 61– Desenvolvendo o programa de importação pelo IDE:

```
// Analise o arquivo CLIENTES.TXT, em sua base de treinamento

#INCLUDE "rwmake.ch"
User Function ImportCli

//Declaracao de Variaveis
Private oLeTxt
Private cString := "SA1"
dbSelectArea("SA1")
dbSetOrder(1)
// Montagem da tela de processamento
@ 200,1 TO 380,380 DIALOG oLeTxt TITLE OemToAnsi("Leitura de Arquivo Texto")
@ 02,10 TO 080,190
@ 10,018 Say " Este programa ira ler o conteudo de um arquivo texto, conforme"
@ 18,018 Say " os parametros definidos pelo usuario, com os registros do arquivo"
@ 26,018 Say " SA1"
@ 70,128 BMPBUTTON TYPE 01 ACTION OkLeTxt()
@ 70,158 BMPBUTTON TYPE 02 ACTION Close(oLeTxt)
```

Coluna R_E_C_D_E_L_

Utilizado na composição da chave única, pois a exclusão de registros ocorre de forma lógica.
Quando um registro é inserido, o conteúdo da coluna R_E_C_D_E_L_ é igual à 0 (zero).
Quando este registro é excluído, a coluna R_E_C_D_E_L_ recebe o conteudo de R_E_C_N_O_, permitindo assim que a chave única não seja invalidade um registro excluído.

Exemplos:

Chave única da tabela – CODIGO

A chave única da tabela será criada no banco desta maneira: CODIGO, R_E_C_D_E_L_.

Momento I – Inserção do registro

CODIGO	R_E_C_N_O_	D_E_L_E_T_	R_E_C_D_E_L
001	1	" "	0

Momento II – Exclusão do registro

CODIGO	R_E_C_N_O_	D_E_L_E_T_	R_E_C_D_E_L
001	1	"* "	1

Momento III – Inclusão da mesma chave

CODIGO	R_E_C_N_O_	D_E_L_E_T_	R_E_C_D_E_L
001	1	"*"	1
001	2	" "	0

Momento IV – Exclusão do registro

CODIGO	R_E_C_N_O_	D_E_L_E_T_	R_E_C_D_E_L
001	1	"*"	1
001	2	"*"	2

Momentos de Flush

A execução do comando de gravação no banco de dados pode não ocorrer no mesmo momento em que é executado um MsUnLock(). Por questões de desempenho, o Protheus faz um cache desses comandos e de tempos em tempos os aplica no banco, porém esta execução pode ser antecipada pelas seguintes ações:

Quando houver um desposicionamento do ponteiro do registro da tabela que teve a inserção;
Quando da execução da função FKCommit().

Escrita do Código

Existem algumas funções que devem ser sempre utilizadas na escrita de um código ADVPL com SQL. Abaixo, serão demonstrados estes comandos.

RetSQLName
Retorna o nome da tabela de acordo com a Empresa em que se está logado.

Sintaxe:
RetSQLName(cAlias)

Parâmetros:
cAlias Alias da tabela. Exemplo: "SA1", "SB1", "SC2", etc.

ChangeQuery
Retorna a query compatibilizada para os bancos de dados homologados para trabalhar com o Protheus.

Sintaxe:
ChangeQuery(cQuery)

Parâmetros:
cQuery Expressão com a instrução SQL.

Anotações

T

CGenQry
Interpreta a instrução SQL e retorna o banco de dados criado.

Sintaxe:
TCGenQry(Nil, Nil, cQuery)

Parâmetros:
cQuery Expressão com a instrução SQL.

GetNextAlias
Retorna o próximo alias disponível para utilização.

Sintaxe:
GetNextAlias()

DBUseArea
Abre um arquivo de dados.

Sintaxe:
DBUseArea(lNewArea, cDriver, cName, xcAlias, lShared, lReadonly)

Parâmetros:

1NewArea - Valor Lógico opcional. Quando verdadeiro (.T.), seleciona o último número ocupado na área de trabalho (WorkArea). Quando falso (.F.) ou omitido, a área corrente é utilizada. Se esta área estiver ocupada, a mesma será fechada antes.

cDriver - Especifica o nome do driver do banco de dados. Na utilização de Query´s em ADVPL, utilizaremos o driver TOPCONN.

cName - Especifica o nome do banco de dados.

xcAlias - Especifica o Alias associado a área de trabalho.

lShared - Especifica se o banco de dados poderá ser acessado por outro processo.

lReadOnly - Define se a área de trabalho será aberta somente para leitura, não permitindo atualizações na mesma.

Anotações

Utilização de Query em Relatório

```
#INCLUDE 'Protheus.CH'
User Function RelSB1()

Local cDesc1  := "Este programa tem como objetivo imprimir relatorio "
Local cDesc2  := "de acordo com parametros informados pelo usuario."
Local cDesc3  := "Listagem de Produtos"
Local cTitulo := "Impressao do Cadastro de Produtos"
Local nLin    := 80
Local cCabec1 := 'CODIGO         DESCRICAO                TIPO  GRUPO'
Local cCabec2 := ""

Private aOrd      := {"CODIGO","DESCRICAO","TIPO","GRUPO"}
Private lEnd      := .F.
Private lAbortPrint := .F.
Private nLimite   := 132
Private cTamanho  := "M"
Private cNomeprog  := "RELSB1"
Private nTipo     := 18
Private aReturn   := { "Zebrado", 1, "Administracao", 2, 2, 1, "", 1}
Private nLastKey  := 0
Private cPerg     := "RELSB1"
Private m_pag     := 01
Private wnrel     := "RELSB1"
Private cString   := "SB1"
AjustaSX1()
Pergunte(cPerg,.F.)

wnrel := SetPrint(cString,cNomeProg,cPerg,@cTitulo,cDesc1,cDesc2,cDesc3,.F.,aOrd,.F.,cTamanho,,.F.)
SetDefault(aReturn,cString)
If nLastKey == 27
  Return
Endif

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

RptStatus({|| RunReport(cCabec1,cCabec2,cTitulo,nLin) },cTitulo)
Return
```

Anotações

```
Static Function RunReport(cCabec1,cCabec2,cTitulo,nLin)
Local cAliasQry := GetNextAlias()
Local cQuery    := ''
Local nOrdem    := aReturn[8]

cQuery += "SELECT SB1.B1_COD, SB1.B1_DESC, SB1.B1_TIPO, SB1.B1_GRUPO "
cQuery += "FROM " + RetSQLName( "SB1" ) + " SB1 "
cQuery += "WHERE "
cQuery += "SB1.B1_FILIAL = '" + xFilial("SB1") + "' AND "
cQuery += "SB1.B1_COD BETWEEN '" + MV_PAR01 + "' AND '" + MV_PAR02 + "' AND "
cQuery += "SB1.B1_TIPO BETWEEN '" + MV_PAR03 + "' AND '" + MV_PAR04 + "' AND "
cQuery += "SB1.B1_GRUPO BETWEEN '" + MV_PAR05 + "' AND '" + MV_PAR06 + "' "
cQuery += " AND D_E_L_E_T_ = '' "

cQuery += "ORDER BY "

If nOrdem == 1 //— Ordem: Codigo
        cQuery += " SB1.B1_FILIAL, SB1.B1_COD"

ElseIf nOrdem == 2 //— Ordem: Descricao
        cQuery += " SB1.B1_FILIAL, SB1.B1_DESC"

ElseIf nOrdem == 3 //— Ordem: Tipo
        cQuery += " SB1.B1_FILIAL, SB1.B1_TIPO"

ElseIf nOrdem == 4 //— Ordem: Grupo
        cQuery += " SB1.B1_FILIAL, SB1.B1_GRUPO"

EndIf

cQuery := ChangeQuery(cQuery)
dbUseArea( .T., "TOPCONN", TCGENQRY(,,cQuery),cAliasQry, .F., .T.)

SetRegua((cAliasQry)->(RecCount()))
While !(cAliasQry)->(EoF())

        If lAbortPrint
                @ 000,000 PSay '***CANCELADO PELO OPERADOR!'
                Exit
        EndIf

        lEndDo

If aReturn[5]==1
  dbCommitAll()
  SET PRINTER TO
  OurSpool(wnrel)
Endif
MS_FLUSH()
Return
```

```
If nLin > 55
                nLin := Cabec( cTitulo, cCabec1, cCabec2, cNomeProg, cTamanho, nTipo )
        EndIf

        nLin++
        @ nLin,000 PSay         (cAliasQry)->B1_COD + Space(02) +;
                                                (cAliasQry)->B1_DESC + Space(03) +;
                                                (cAliasQry)->B1_TIPO + Space(04) +;
                                                (cAliasQry)->B1_GRUPO

        (cAliasQry)->(DbSkip())
        IncRegua()
If aReturn[5]==1
  dbCommitAll()
  SET PRINTER TO
  OurSpool(wnrel)
Endif
MS_FLUSH()
Return


Static Function AjustaSX1()
PutSx1(         "RELSB1","01","Do Produto:","","","";
                "mv_ch1","C",15,0,1,"G","","","","","","mv_par01")

PutSx1(         "RELSB1","02","Ate o Produto:","","","";
                "mv_ch2","C",15,0,1,"G","","","","","","mv_par02")

PutSx1(         "RELSB1","03","Do Tipo:","","","";
                "mv_ch3","C",02,0,1,"G","","","","","","mv_par03")

PutSx1(         "RELSB1","04","Ate o Tipo:","","","";
                "mv_ch4","C",02,0,1,"G","","","","","","mv_par04")

PutSx1(         "RELSB1","05","Do Grupo:","","","";
                "mv_ch5","C",04,0,1,"G","","","","","","mv_par05")

PutSx1(         "RELSB1","06","Ate o Grupo:","","","";
                "mv_ch6","C",04,0,1,"G","","","","","","mv_par06")

Return .T.
```

Anotações

```
#Include 'Protheus.CH'
User Function MyDbTree()

Local oDlg     := Nil
Local oDlgTree  := Nil
Local oTree    := Nil
Local oFont    := Nil
Local nTop     := oMainWnd:nTop+23
Local nLeft    := oMainWnd:nLeft+5
Local nBottom   := oMainWnd:nBottom-60
Local nRight    := oMainWnd:nRight-10
Local nOpc     := 0
Local cCliente  := CriaVar('A1_COD')
Local cLoja    := CriaVar('A1_LOJA')
Local cSocial   := Space(30)
Local aObjs    := Array(6)
Local nAux     := 0

Private aHeaderSD2 := {}
Private aColsSD2   := {}

Private aHeaderSC6 := {}
Private aColsSC6   := {}

Private aRotina := {       { ,,0,1},;
                                  { ,,0,2},;
                                  { ,,0,3},;
                                  { ,,0,4},;
                                  { ,,0,5} }
INCLUI := .F.
ALTERA := .F.

//— Montagem do aHeader dos Itens da Nota Fiscal
dbSelectArea("SX3")
dbSetOrder(1)
```

Anotações


```
//— Montagem do aHeader dos Itens dos Pedidos de Venda
MsSeek('SC6')
While !EOF() .And. (X3_ARQUIVO == "SC6")
        IF X3USO(X3_USADO) .And. cNivel >= X3_NIVEL
                AADD(aHeaderSC6,{ TRIM(X3Titulo()), X3_CAMPO, X3_PICTURE,;
                                X3_TAMANHO, X3_DECIMAL, X3_VALID,;
                                X3_USADO, X3_TIPO, X3_ARQUIVO,X3_CONTEXT } )
        EndIf
        dbSkip()
EndDo
Aadd(aColsSC6,Array(Len(aHeaderSC6)+1))

//— Montagem do aHeader dos Itens da Nota Fiscal
MsSeek("SD2")
While !EOF() .And. (X3_ARQUIVO == "SD2")
        IF X3USO(X3_USADO) .And. cNivel >= X3_NIVEL
                AADD(aHeaderSD2,{ TRIM(X3Titulo()), X3_CAMPO, X3_PICTURE,;
                                X3_TAMANHO, X3_DECIMAL, X3_VALID,;
                                X3_USADO, X3_TIPO, X3_ARQUIVO,X3_CONTEXT } )
        EndIf
        dbSkip()
EndDo
Aadd(aColsSD2,Array(Len(aHeaderSD2)+1))

//— Preenchimento dos aCols
For nAux := 1 to Len(aHeaderSC6)
        aColsSC6[1][nAux] := CriaVar(aHeaderSC6[nAux][2])
        aColsSC6[1][Len(aHeaderSC6)+1] := .F.
Next

//— Preenchimento dos aCols
For nAux := 1 to Len(aHeaderSD2)
        aColsSD2[1][nAux] := CriaVar(aHeaderSD2[nAux][2])
        aColsSD2[1][Len(aHeaderSD2)+1] := .F.
Next
```

Anotações

```
DEFINE MSDIALOG oDlg TITLE 'My DbTree' OF oMainWnd PIXEL FROM 000,000 TO 100,400
@ 020,005 Say 'Cliente:' SIZE 100,007 OF oDlg PIXEL
@ 020,057 Say 'Loja:' SIZE 050,007 OF oDlg PIXEL
@ 020,077 Say 'R. Social' SIZE 100,007 OF oDlg PIXEL
@ 030,005 MsGet cCliente PICTURE "@!" F3 'SA1' VALID CheckCli(cCliente,,@cSocial) SIZE 050,010 OF oDlg
PIXEL
@ 030,057 MsGet cLoja PICTURE "@!" VALID CheckCli(cCliente,cLoja,@cSocial) SIZE 020,010 OF oDlg PIXEL
@ 030,077 MsGet cSocial PICTURE "@!" WHEN .F. SIZE 100,010 OF oDlg PIXEL


ACTIVATE MSDIALOG oDlg ON INIT EnchoiceBar(oDlg,{|| nOpc := 1, oDlg:End() }, {|| nOpc := 0, oDlg:End() })
CENTERED

SA1->(DbSetOrder(1))
If !SA1->(MsSeek(xFilial('SA1') + cCliente + cLoja ))
   nOpc := 0
EndIf

If nOpc == 1
        DEFINE FONT oFont NAME "Arial" SIZE 0, -10
        DEFINE MSDIALOG oDlgTree TITLE 'My DbTree' OF oMainWnd PIXEL FROM nTop,nLeft TO
nBottom,nRight

        oPanel := TPanel():New(15,160,'',oDlgTree, oDlgTree:oFont, .T., .T.,,,(nRight-nLeft)/2-160,((nBottom-
nTop)/2)-25,.T.,.T. )

        oTree := DbTree():New(15, 2,((nBottom-nTop)/2)-5,159,oDlgTree,,,.T.)
oTree:LShowHint := .F.
        oTree:oFont := oFont
        oTree:bChange := {|| ShowScreen(oTree,aObjs,@oPanel,{ 0,0,((nBottom-nTop)/2)-24,(nRight-
nLeft)/2-160}) }

//— Enchoice com o Cadastro do Cliente
        RegToMemory("SA1",.F.,.T.,.F.)
        aObjs[1] := MsMGet():New("SA1",SA1->(RecNo()),2,,,,,{0,0,((nBottom-nTop)/2)-25,(nRight-nLeft)/2-
160},,3,,,,oPanel,,.T.)

//— Pedidos de Venda
        RegToMemory('SC5',.F.)
        aObjs[2]        :=        MsMGet():New("SC5",SC5->(RecNo()),2,,,,,{0,0,((nBottom-nTop)/4),(nRight-
nLeft)/2},,3,,,,oPanel,,.T.)
        aObjs[2]:Hide()

        //— Itens dos Pedidos de Venda
        aObjs[3] := MsNewGetDados():New(      (nBottom-nTop)/4,;
                                              0,;
                                              (nBottom-nTop)/2-30,;
                                              (nRight-nLeft)/2-160,;
                                              Nil,;
                                              "Allwaystrue","Allwaystrue",,,,,,,,;
                                              oPanel,;
```

```
aHeaderSC6,;
                                        aColsSC6)
                                        aObjs[3]:Hide()

//— Notas Fiscais de Saida
        RegToMemory("SF2",.F.)
        aObjs[4]        :=      MsMGet():New("SF2",SF2->(RecNo()),2,,,,,{0,0,((nBottom-nTop)/4),(nRight-
nLeft)/2},,3,,,,oPanel,,.T.)
        aObjs[4]:Hide()

        //— Itens da Nota Fiscal
        aObjs[5] := MsNewGetDados():New(    (nBottom-nTop)/4,;
                                        0,;
                                        (nBottom-nTop)/2-30,;
                                        (nRight-nLeft)/2-160,;
                                        Nil,;
                                        "Allwaystrue","Allwaystrue",,,,,,,,;
                                        oPanel,;
                                        aHeaderSD2,;
                                        aColsSD2)
                                        aObjs[5]:Hide()
//— Titulos a Receber
        RegToMemory('SE1',.F.)
        aObjs[6] := MsMGet():New("SE1",SE1->(RecNo()),2,,,,,{0,0,((nBottom-nTop)/2)-25,(nRight-nLeft)/2-
160},,3,,,,oPanel,,.T.)
        aObjs[6]:Hide()

//— Chama a rotina de construcao do Tree
        Processa( { || MontaTree(oTree) }, 'Construindo o Tree...' )

        ACTIVATE MSDIALOG oDlgTree ON INIT EnchoiceBar(oDlgTree,{|| oDlgTree:End() },{|| oDlgTree:End()
}) CENTERED
EndIf

//FUNCAO RESPONSAVEL PELA MONTAGEM DO TREE
Static Function MontaTree( oTree )
Local cQuery    := ''
Local cAliasQry := GetNextAlias()
Local cBMP      := ''

oTree:AddTree('Dados Cadastrais: ' + SA1->(A1_COD+A1_LOJA),.T.,,,'FOLDER5','FOLDER6',"SA1"+StrZero(
SA1->(RecNo()),12))

//- Pedidos de Venda do Cliente
oTree:AddTree('Pedidos de Venda',.T.,,,'FOLDER5','FOLDER6',Space(15))
cQuery := "SELECT C5_NUM, C5_CLIENTE, C5_LOJACLI, R_E_C_N_O_ SC5RecNo "
cQuery += "FROM " + RetSqlName("SC5") + " SC5 "
cQuery += "WHERE C5_FILIAL = '" + xFilial('SC5') + "' AND "
cQuery += "C5_CLIENTE = '" + SA1->A1_COD +"' AND "
cQuery += "C5_LOJACLI = '" + SA1->A1_LOJA +"' AND "
cQuery += "D_E_L_E_T_ = ' ' "
cQuery := ChangeQuery(cQuery)
```

```
dbUseArea( .T.,"TOPCONN",TCGENQRY(,,cQuery),cAliasQry, .F., .T.)
While (cAliasQry)->(!Eof())
	oTree:AddTreeItem('Pedido:  '  +  (cAliasQry)->C5_NUM,,,"SC5"+StrZero((cAliasQry)-
>(SC5RecNo),12))
	(cAliasQry)->(DbSkip())
EndDo
(cAliasQry)->(DbCloseArea())
oTree:EndTree()

//- Notas Fiscais do Cliente
oTree:AddTree('Notas Fiscais',.T.,,,'FOLDER5','FOLDER6',Space(15))
cQuery := "SELECT F2_CLIENTE, F2_LOJA, F2_DOC, F2_SERIE, R_E_C_N_O_ SF2RecNo "
cQuery += "FROM " + RetSqlName("SF2") + " SF2 "
cQuery += "WHERE F2_FILIAL = '" + xFilial('SF2') + "' AND "
cQuery += "F2_CLIENTE = '" + SA1->A1_COD +"' AND "
cQuery += "F2_LOJA = '" + SA1->A1_LOJA +"' AND "
cQuery += "D_E_L_E_T_ = ' ' "
cQuery := ChangeQuery(cQuery)

dbUseArea( .T.,"TOPCONN",TCGENQRY(,,cQuery),cAliasQry, .F., .T.)
While (cAliasQry)->(!Eof())
	oTree:AddTreeItem('Nota: ' + (cAliasQry)->(F2_DOC + '/' + F2_SERIE),,,"SF2"+StrZero((cAliasQry)-
>(SF2RecNo),12))
	(cAliasQry)->(DbSkip())
EndDo
(cAliasQry)->(DbCloseArea())
oTree:EndTree()


//— Titulos para o Fornecedor
oTree:AddTree('Titulos a Receber',.T.,,,'FOLDER5','FOLDER6',Space(15))
cQuery := "SELECT E1_CLIENTE, E1_LOJA, E1_PREFIXO, E1_NUM, E1_PARCELA, E1_SALDO, R_E_C_N_O_
SE1RecNo "
cQuery += "FROM " + RetSqlName('SE1') + " SE1 "
cQuery += "WHERE E1_FILIAL = '" + xFilial('SE1') + "' AND "
cQuery += "E1_CLIENTE = '" + SA1->A1_COD +"' AND "
cQuery += "E1_LOJA = '" + SA1->A1_LOJA +"' AND "
cQuery += "D_E_L_E_T_ = ' ' "
cQuery := ChangeQuery(cQuery)

dbUseArea( .T.,"TOPCONN",TCGENQRY(,,cQuery),cAliasQry, .F., .T.)
While (cAliasQry)->(!Eof())
	If (cAliasQry)->E1_SALDO == 0
		cBMP := 'BR_VERMELHO_OCEAN'
	Else
		cBMP := 'BR_VERDE_OCEAN'
	EndIf

	oTree:AddTreeItem('Titulo: ' + (cAliasQry)->(E1_PREFIXO+'/'+E1_NUM+'/'+E1_PARCELA),cBMP,cB
MP,"SE1"+StrZero((cAliasQry)->(SE1RecNo),12))
	(cAliasQry)->(DbSkip())
EndDo
```

```
(cAliasQry)->(DbCloseArea())
oTree:EndTree()

//— Fecha o Tree Principal
oTree:EndTree()

Return

//FUNCAO RESPONSAVEL PELA EXIBICAO DOS OBJETOS CONFORME SELECAO NO TREE
Static Function ShowScreen(oTree,aObjs,oPanel,aPos)
Local cAlias       := SubStr(oTree:GetCargo(),1,3)
Local nRecView := Val(SubStr(oTree:GetCargo(),4,12))
Local nCnt     := 0
Local nObjs    := 0
Local nAux     := 0
aCols  := {}

//— Esconde as enchoices/getdados
For nCnt := 1 To Len(aObjs)
        If ValType(aObjs[nCnt]) == "O"
                aObjs[nCnt]:Hide()
        EndIf
Next nCnt

If nRecView <> 0
        dbSelectArea(cAlias)
        MsGoto(nRecView)

        If cAlias == 'SA1'
                RegToMemory("SA1",.F.,.T.,.F.)
                nObjs := 1

        ElseIf cAlias == 'SC5'
                RegtoMemory(cAlias,.F.)
                nObjs := 2
                aColsSC6 := {}

                SC6->(DbSetOrder(1)) //C6_FILIAL+C6_NUM+C6_ITEM+C6_PRODUTO
                SC6->(MsSeek(xFilial('SC6') + SC5->C5_NUM))
                While SC6->(C6_FILIAL+C6_NUM) == xFilial('SC6') + SC5->C5_NUM
                        Aadd(aColsSC6,Array(Len(aHeaderSC6)+1))
                        For nAux := 1 To Len(aHeaderSC6)
                                If      aHeaderSC6[nAux,10] != "V""
                                        aColsSC6[Len(aColsSC6),nAux] := SC6->(FieldGet(FieldPos(aHead
erSC6[nAux,2])))

Else
                                        aColsSC6[Len(aColsSC6),nAux] := CriaVar(aHeaderSC6[nAux,2])
                                EndIf
                        Next
                        aColsSC6[Len(aColsSC6),Len(aHeaderSC6)+1] := .F.
                        SC6->(dbSkip())
```

```
                EndDo
                aObjs[3]:aCols := AClone(aColsSC6)
                aObjs[3]:Refresh(.T.)
                aObjs[3]:Show()

        ElseIf cAlias == "SF2"
                RegtoMemory(cAlias,.F.)
                nObjs := 4
                aColsSD2 := {}

                SD2->(DbSetOrder(3))     //D2_FILIAL+D2_DOC+D2_SERIE+D2_CLIENTE+D2_LOJA+D2_
COD+D2_ITEM
                SD2->(MsSeek(xFilial('SD2') + SF2->(F2_DOC+F2_SERIE+F2_CLIENTE+F2_LOJA)))
                While SD2->(D2_FILIAL+D2_DOC+D2_SERIE+D2_CLIENTE+D2_LOJA) == xFilial('SD2') +
SF2->(F2_DOC+F2_SERIE+F2_CLIENTE+F2_LOJA)
                        Aadd(aColsSD2,Array(Len(aHeaderSD2)+1))
                        For nAux := 1 To Len(aHeaderSD2)
                                If      aHeaderSD2[nAux,10] != "V"
                                        aColsSD2[Len(aColsSD2),nAux] := SD2->(FieldGet(FieldPos(aHea
derSD2[nAux,2])))
                                Else
                                        aColsSD2[Len(aColsSD2),nAux] := CriaVar(aHeaderSD2[nAux,2])
                                EndIf
                        Next
                        aColsSD2[Len(aColsSD2),Len(aHeaderSD2)+1] := .F.
                        SD2->(dbSkip())
                EndDo
                aObjs[5]:aCols := AClone(aColsSD2)
                aObjs[5]:Refresh(.T.)
                aObjs[5]:Show()

        ElseIf cAlias == 'SE1'
                RegToMemory(cAlias,.F.)
                nObjs := 6

        EndIf

aObjs[nObjs]:Refresh(.T.)
        aObjs[nObjs]:Show()
EndIf
Return

Static Function CheckCli(cCli, cLoja, cSocial)
Default cLoja := '01'

SA1->(DbSetOrder(1))
If SA1->(MsSeek(xFilial('SA1') + cCli + cLoja))
        cSocial := SA1->A1_NOME
Else
        cSocial := 'CLIENTE NAO ENCONTRADO!'
EndIf
Return( .T. )
```

IMPORTAÇÃO DE DADOS

O que você irá aprender neste capítulo

Neste capítulo, iremos abordar as customizações feitas pelo IDE, para importar dados para tabelas.

Rotinas abordadas

• Criando o Programa de Importação;

Exercício 61– Desenvolvendo o programa de importação pelo IDE:

```
// Analise o arquivo CLIENTES.TXT, em sua base de treinamento
#INCLUDE "rwmake.ch"
User Function ImportCli

//Declaracao de Variaveis
Private oLeTxt
Private cString := "SA1"
dbSelectArea("SA1")
dbSetOrder(1)
// Montagem da tela de processamento

@ 200,1 TO 380,380 DIALOG oLeTxt TITLE OemToAnsi("Leitura de Arquivo Texto")
@ 02,10 TO 080,190
@ 10,018 Say " Este programa ira ler o conteudo de um arquivo texto, conforme"
@ 18,018 Say " os parametros definidos pelo usuario, com os registros do arquivo"
@ 26,018 Say " SA1"
@ 70,128 BMPBUTTON TYPE 01 ACTION OkLeTxt()
@ 70,158 BMPBUTTON TYPE 02 ACTION Close(oLeTxt)
```

Anotações

PONTOS DE ENTRADAS

O que você irá aprender neste capítulo

Neste capítulo, aprenderemos a criar e implementar Pontos de Entradas, junto as rotinas padrões do sistemas.

Rotinas Abordadas

• Ferramentas de Auxílio na implementação de Pontos de Entradas (Quark);
• Criação de Pontos de Entradas;
• Implementação de Pontos de Entradas, junto as rotinas padrões.

Pontos de Entrada

O conceito utilizado para a Criação de um Ponto de Entrada é o mesmo da herança adotado nas linguagens para Banco de Dados, ou seja, (Store Procedures).

São chamadas de Programas colocadas em pontos estratégicos nas funções padrões do sistema e que originalmente não fazem nada.

Assim que o desenvolvedor identificar a necessidade de uma intervenção nestes pontos, basta criar a rotina, dar a ela o nome específico citado na documentação do Protheus e compilá-la.

No momento em que a Rotina Padrão for disparada e passar pela customização do Ponto de Entrada, o mesmo também será executado.

Podemos dizer que esta é a maneira mais prática de intervenção aos Programas Padrões do sistema sem a necessidade de alteração dos mesmos, fazendo com que estas implementações passem despercebidas pelo usuário no momento de sua execução.

Porém é necessário que saibamos muito bem sobre a Lógica do Programa em questão e até mesmo qual é a situação de memória utilizada naquele momento.

Exercício 62– Como Identificar os Nomes dos Pontos de Entradas:

• Desenvolva um "Ponto de Entrada", para o "Cadastro de Clientes", sendo que o mesmo dispare uma "Mensagem Padrão", após a "Confirmação do Cadastro".

1. Através de um "Editor de Textos", edite o menu do "Módulo de Faturamento" e anote o "Nome da Função", utilizada para o "Cadastro de Clientes";

2. Na seqüência, execute a "Ferramenta – Quark.exe", que se encontra na pasta "AP7\Rdmake\Quark";

3. Acesse a opção "Pontos de Entrada", pressione as teclas "<Alt> + P", selecione a opção "Pesquisar por Programa" e informe o "Nome do Programa", responsável pelo "Cadastro de Clientes";

4. Assim que o sistema posicionar sobre os "Pontos de Entrada", disponíveis para essa rotina, anote o "Nome do Responsável", pela execução após a "Inclusão do Cliente" e saia da "Ferramenta – Quark".

Observação:

• A "Ferramenta – Quark", utilizada em curso é de uso interno da MICROSIGA e pelo suporte externo, para a utilização deste recurso será necessário acessar o site que contém todas as informações disponibilizadas para clientes e apoio à "Consultorias" no seguinte endereço: "WWW.MICROSIGA.COM.BR", acessando o link "Utilidades" + "Quark";

• É importante lembrar que para acessá-lo é necessário o uso do "SenhaP".

Exercício 63 – Como Criar Pontos de Entrada:

1. Acesse a "Ferramenta – IDE";

2. Crie o seguinte "Programa":

```
#Include "Rwmake.ch"
User Function m030inc()
MsgBox("O Cliente foi cadastrado com sucesso!")
Return
```

3. Confira os dados e salve o "Programa" na pasta "Rdmake" com o mesmo nome dado a "Função", para que fique fácil a sua identificação posteriormente;

4. Compile o "Programa" e saia da "Ferramenta – IDE".

Exercício 64 – Como Testar Pontos de Entrada:

1. Acesse o "Módulo de Faturamento";

2. Selecione as seguintes opções "Atualização" + "Cadastros" + "Clientes";

3. Clique na opção "Incluir" e informe os dados a seguir:

Código:	000002
Loja:	01
Nome:	Cliente 02
N Fantasia:	Cliente 02
Tipo:	R=Revendedor
Endereço:	Avenida Braz Leme, 1399
Município:	São Paulo
Estado:	SP

4. Confira os dados, confirme o cadastro de "Clientes" e verifique que neste momento o "Ponto de Entrada" será disparado, trazendo a mensagem definida no programa;

5. Cancele o próximo cadastro e saia do "Módulo de Faturamento".

Exercício 65 – Como criar outro Ponto de Entrada:

• Desenvolva um "Ponto de Entrada", para o "Cadastro de Clientes", sendo que o mesmo dispare uma "Mensagem Padrão", após a "Exclusão" de um registro no "Cadastro de Clientes".

1. Execute os procedimentos utilizados no exercício anterior, para identificar o "Nome do Ponto de Entrada", que se enquadra nesta customização;

2. Na seqüência, acesse a "Ferramenta – IDE e crie o seguinte programa:

```
#Include "Rwmake.ch"
User Function m030exc()
MsgBox("O Cliente foi excluido com sucesso!")
Return
```

3. Confira os dados, salve o programa na pasta "Rdmake", com o mesmo nome dado a "Função", para que fique fácil a sua identificação posteriormente;

4. Compile o "Programa" e saia da "Ferramenta – IDE".

Exercício 66 C omo Testar Pontos de Entradas:

1. Acesse o "Módulo de Faturamento";

2. Selecione as seguintes opções "Atualização" + "Cadastros" + "Clientes";

3. Posicione com o cursor sobre o "Cliente – 000002", clique na opção "Excluir" e verifique que neste momento o "Ponto de Entrada" será disparado, trazendo a mensagem definida no programa

4. Confira os dados e abandone o "Módulo de Faturamento".

Exercício 67 – Como Criar outro Ponto de Entrada:

• Desenvolva um "Ponto de Entrada" para o "Cadastro de Produtos", sendo que no momento da "Confirmação do Cadastro", a rotina dispare uma "Atualização dos Dados", para o "Cadastro de Complementos de Produtos".

1. Execute os procedimentos utilizados no exercício anterior, para identificar o "Nome do Ponto de Entrada", que se enquadra nesta customização;

2. Na seqüência, acesse a "Ferramenta – IDE" e crie o programa a seguir:

```
#Include "Rwmake.ch"
User Function mt010inc()
Local aArea := GetArea()
If MsgBox("Confirma a Geração do Complemento","Complemento de Produtos",;
                "YESNO")
DbSelectArea("SB5")
If RecLock("SB5",.T.)
        SB5->B5_FILIAL:= SB1->B1_FILIAL
        SB5->B5_COD   := SB1->B1_COD
        SB5->B5_CEME := SB1->B1_DESC
Else
        MsgAlert("Não foi possivel travar o registro!")
EndIf
```

```
        MsUnLock("SB5")
EndIf
RestArea(aArea)
Return
```

3. Confira os dados, salve o programa na pasta "Rdmake", com o mesmo nome dado a "Função", para que fique fácil a sua identificação posteriormente;

Código:	000003
Descrição:	PRODUTO 3
Tipo:	PA
Unidade:	CX
Armazém Pad.:	01

4. Compile o "Programa" e saia da "Ferramenta – IDE".

Exercício 68 – Como Testar o Ponto de Entrada:

1. Acesse o "Módulo de Faturamento";

2. Selecione as seguintes opções "Atualização" + "Cadastros" + "Produtos";

3. Clique na opção "Incluir" e informe os dados a seguir:

4. Confira os dados, confirme o cadastro de "Produtos" e verifique que neste momento o "Ponto de Entrada" será disparado, trazendo a mensagem definida no programa;

5. Na seqüência, saia do "Cadastro de Produtos", acesse a opção "Complem. Produtos" e verifique que os "Dados do Produto" cadastrado anteriormente, foram atualizados também neste outro arquivo;

6. Saia do "Cadastro de Complemento de Produtos" e do "Módulo de Faturamento".

Funções para o interpretador xbase / advpl

A seguir são apresentadas as funções SIGA MP8 para uso junto ao RD-MAKE / Interpretador xBASE.
Na linha seguinte ao nome de cada função, é informado onde normalmente ela é utilizada, a saber:

• Processamento: funções usadas em cálculos, acesso a arquivos e tratamentos em geral;
· Impressão: funções usadas exclusivamente na geração de relatórios;
· Telas: funções usadas na geração de telas, seja DOS ou Windows;

AbreExcl

Tipo: Processamento

Fecha o arquivo e reabre exclusivo. Esta função fecha o arquivo cujo alias está expresso em <cAlias> e reabre-o em modo exclusivo para proceder operações em que isto é necessário, como por exemplo, PACK.

Entretanto, é preferível utilizar o depurador do sistema para proceder estas operações. Se outra estação estiver usando o arquivo, o retorno será .F..

Sintaxe

AbreExcl(cAlias)

Parâmetros

cAlias – Nome do Alias do Arquivo. Deve ter obrigatóriamente sua estrutura definida no SX3.

Exemplo:

```
IF AbreExcl("SI2")
Pack
ENDIF
AbreExcl( )
dbGoTop( )
```

Anotações

Activate Dialog

Tipo: Tela Windows

Ativa uma janela previamente definida na função Dialog e executa os GETs, botões e outros objetos.

Sintaxe

ACTIVATE DIALOG cVar <CENTERED> [On Init cFuncInit] [Valid cFuncValid]

Parâmetros

cVar – Variável utilizada na função Dialog para definição da janela.

cFuncInit – Função executada automaticamente na abertura do diálogo na tela (Opcional).

cFuncValid– Função executada para validar o fechamento da janela de diálogo. Deve retornar um valor lógico (.T. ou .F.) (Opcional)

Comentários

A cláusula <CENTERED> é opcional, se omitida assume as coordenadas definidas na criação da janela.

Exemplo:

Ver exemplo no programa RDDEMO apresentado no final deste Manual.

Ver também

Função Dialog

Anotações

AADD()

Adiciona um novo elemento ao final do Vetor

Sintaxe:

aAdd(aDestino,expValor)
aDestino - Vetor ao qual será armazenado o valor
expValor - Valor Atribuido ao novo Vetor.

AllTrin()

Remove os espaços em branco a direita e esquerda de uma cadeia de caracteres.

ALLTRIM(cString)

cString - expressão caractere cujos os espaços em branco serão eliminados

ASCAN()

Varre um vetor procurando um valor ou at, que um bloco retorne verdadeiro (.T.)

Sintaxe

ASCAN(<aDestino>, <ProcuraExp>,
[<nInicio>], [<nCont>]) --> nParouEm
Argumentos
<aDestino> , o vetor a ser varrido.

<ProcuraExp> pode ser um valor simples a ser procurado, ou um bloco de código. Caso <ProcuraExp> seja um valor simples, este poder ser do tipo numerico, lógico, data, ou caractere.

<nInicio> , o elemento a partir do qual ter início a pesquisa. Se este argumento nao for especificado, a posição inicial padrão , um.

<nCont>, a quantidade de elementos que serao varridos a partir da posição inicial. Caso este argumento nao seja especificado, todos os elementos, desde o elemento inicial até, o final do vetor, serão varridos.

Retorno

ASCAN() retorna um valor numérico que representa a posição ocupada no vetor pelo último elemento varrido. Se <ProcuraExp> for um valor simples, ASCAN() retorna a posição do primeiro elemento que corresponder ao valor procurado, ou zero caso nao haja correspondência. Se <ProcuraExp> for um bloco de código, ASCAN() retorna a posição do elemento onde o bloco retornou verdadeiro (.T.).

Anotações

Aleatorio

Tipo: Processamento

Gera um número aleatório de acordo com a semente passada.

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

Sintaxe

Aleatorio(nMax,nSeed)

Parâmetros

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

Retorna

nRet – Número aleatório retornado

Exemplo:

// Exemplo do uso da função Aleatorio:

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

```
Next i
inkey(0)
Return
```

Anotações

Avalimp

Tipo: Relatórios

Configura a impressora por meio dos parâmetros. Esta função é usada em relatórios específicos que não se utilizam da função "Cabec".

Imprimindo o retorno desta função na impressora, ela se encarregará de configurar a impressora de acordo com o arquivo de driver escolhido e com as configurações escolhidas pelo usuário definidas no array aReturn.

Sintaxe

AvalImp(nLimit)

Parâmetros

nLimit – Tamanho do relatório em colunas. Pode ser 80, 132 ou 220 (respec-tivamente para relatórios de tamanho "P","M" e "G").

Retorna

cControl – String com caracteres de controle, dependente das configurações escolhidas pelo usuário e do arquivo de driver especificado.

Exemplo:

```
// Exemplo de uso da função AvalImp:
#IFNDEF WINDOWS
#DEFINE PSAY SAY
#ENDIF
cCbTxt:= ""
cCbCont:= ""
nOrdem:= 0
nAlfa:= 0
nZ:= 0
nM:= 0
cTamanho:= "G"
cLimite:= 220
cTitulo:= PADC("Nota Fiscal",74)
```

```
cDesc1:= PADC("Este programa irá emitir a Nota Fiscal de Entrada/Saída",74)
cDesc2:= ""
cDesc3:= PADC("da Feeder Industrial Ltda.",74)
cNatureza:= ""
aReturn:= {"Especial", 1,"Administração", 1, 2, 2,"",1}
cNomeProg:= "NFEEDER"
cPerg:= "ENTSAI"
nLastKey:= 0
lContinua:= .T.
nLi:= 0
wnrel:= "NFEEDER"
nTamNf:=72 // Apenas Informativo
Pergunte(cPerg,.F.) // Pergunta no SX1
cString:="SF2"
wnrel:= SetPrint(cString,wnrel,cPerg,cTitulo,cDesc1,cDesc2,cDesc3,.T.)
SetDefault(aReturn,cString)
If nLastKey == 27
Return
Endif
#IFDEF WINDOWS
RptStatus({|| Execute(Relato)})
Return

#ENDIF

Function Relato
SetPrc(0,0)

// Aqui está a chamada da função AvalImp. Configura a
// impressora de acordo com as definições em aReturn
// escolhidas pelo usuário na função SetPrint
@ 00,00 PSAY AvalImp(220)
dbSelectArea("SF2")
dbSeek(xFilial()+mv_par01+mv_par03,.T.)
// O programa segue normalmente...
Return
```

Anotações

ALIAS()

Retorna o alias de uma rea de trabalho especificada

Sintaxe

ALIAS([<nAreaTrabalho>]) --> cAlias

Argumentos

<nAreaTrabalho>, o n£mero de qualquer rea de trabalho.

Retorno

ALIAS() retorna o alias da área de trabalho especificada na forma de uma cadeia de caracteres. Caso <nAreaTrabalho> nao seja especificada, retornado o alias da área de trabalho corrente. Se nao houver nenhum arquivo de banco de dados em USo na área de trabalho especificada, ALIAS() retorna uma cadeia de caracteres nula ("").

Anotações

Aviso

Tipo: Tela DOS/Windows

Monta uma janela exibindo o texto desejado e, opcionalmente, disponibilizando opções de escolha para o usuário.

Sintaxe

Aviso(cTitulo,cMensagem,aOpcoes)

Parâmetros

cTitulo – Titulo da janela.

cMensagem – Mensagem para ser exibida no interior da janela. O tamanho máximo é de 90 caracteres.
aOpcoes – Array de caracteres com as opções para a montagem de menu (na versão DOS) ou dos botões (na versão Windows).
Retorna

nResp – Retorno. Retorna o número da opção escolhida pelo usuário.

Exemplo:

```
// Exemplo de uso da função Aviso:
While .T.
GravaArq() // Função qualquer (apenas p/exemplo)
If !File("TESTE.TXT")
aOp:= {"Sim","Nao","Cancela"}
cTit:= "Atencao!"
cMsg:= "O arquivo TESTE.TXT nao foi gravado!"
cMsg:= cMsg + " Tenta novamente?"
nOp:= Aviso(cTit,cMsg,aOp)
If nOp == 1 // Sim
```

```
Loop
ElseIf nOp == 3 // Cancela
Return
Else // Nao ou <ESC>
Exit
Endif
Endif
Exit
EndDo
// Faz o processamento...
Return
```

Anotações

@ n1,n2 BmpButton

Tipo: Tela Windows

Cria um botão de bitmap padrão do SigaAdv Win.

Sintaxe

@ nLinha,nColuna BMPBUTTON TYPE nBotao ACTION cFuncao OBJECT oBtn

Parâmetros

nLinha – Número da linha superior
nColuna – Número da coluna superior
nBotao – Número do botão padronizado
cFuncao – Função que será executada
oBtn – Objeto associado ao botão

Comentários

Para executar funções definidas em um mesmo .PR? utilizar a função

Execute("Nome da função") ou ExecBlock("Nome do Arquivo") para chamar outro .PR?.

Exemplo:

Ver exemplo no programa RDDEMO apresentado no final deste Manual.

@...To...Browse

Tipo: Tela Windows

Ativa Browse padrão SigaAdv Win.

Sintaxe

@ nLinha1,nColuna1 TO nLInha2,nColuna2 BROWSE cAlias <ENABLE> cCor

nLinha1 – Número da linha superior
nColuna1 – Número da coluna superior
nLinha2 – Número da linha inferior
nColuna2 – Número da coluna inferior

cAlias – Alias do Arquivo (apenas arquivos com estrutura no SX3)
cCor – Expressão que identifica a cor da marca do registro (opcional)

Exemplo:

Marca "Verde" - Titulo em aberto
Marca "Vermelha" - Titulo pago

Comentários

A cláusula <ENABLE> é opcional, se for omitida não será disponibilizada coluna que identifica situação do registro (Cor verde/vermelha).

Anotações

@...Button

Tipo: Tela Windows

Cria um botão com texto.

Sintaxe

@ nLinha,nColuna BUTTON cTexto SIZE nAltura,nLargura ACTION cFunção Object oBtn

Parâmetros

nLinha – Número da linha superior
nColuna – Número da coluna superior

cTexto – Texto que será apresentado no botão. Deve incluir um "_" antes
da letra que utilizada como Hot Key. Ex.: ("_Salvar","Edi_Tar")
nAltura – Altura do botão
nLargura – Largura do botão
cFunção – Função que será executada
Object oBtn – Objeto associado ao botão.

Comentários

Para executar funções definidas em um mesmo .PR? utilizar a função Execute("Nome da função") ou ExecBlock("Nome do Arquivo") para chamar outro .PR?.

Exemplo:

Ver exemplo no programa RDDEMO apresentado no final deste Manual.

Anotações

Cabec

Tipo: Impressão

Esta função imprime, na impressora selecionada, o cabeçalho padrão dos relatórios do SIGA MP8. Devolve o número da última linha impressa para que seja dada continuidade ao relatório.

Sintaxe

Cabec(cTítulo, cTexto1, cTexto2, cProg, cLargura, cControle)

Parâmetros

cTítulo – Título do Relatório
cTexto1 – Extenso da primeira linha do cabeçalho
cTexto2 – Extenso da segunda linha do cabeçalho
cProg – Nome do Programa
cLargura – Largura do relatório (P/ M/ G)
cControle – Caractere de controle da impressora (numérico)

Retorna

nLinha – Número da última linha impressa no cabeçalho

Exemplo:

cTitulo := "Relação dos Adiantamentos"
cCabec1 := "Código Item Conta Contábil CCusto Projeto Data Valor"

```
cCabec2 := "———————— ————— ————————————————— ————————— ————————— ————————
————————"
cPrograma := "ATRF090"
cTamanho := "P"
nCaracter := 15
cRel:=SetPrint(cAlias, cPrograma , , @cTitulo, cDesc1, cDesc2, cDesc3 , .T., aOrd )
SetDefault(aReturn, cString)
nLinha:=Cabec(cTitulo, cCabec1, cCabec2, cPrograma, cTamanho, nCaracter)
While !EOF()
nLinha:=nLinha+1
@nLinha,1 Say SB1->B1_CODIGO
```

CGC

Tipo: Processamento

Consiste o CGC digitado, tomando como base o algoritmo nacional para verificação do dígito de controle.

Esta função procede um cálculo do dígito verificador do número do Cadastro Geral de Contribuintes do Ministério da Fazenda. É utilizado o dígito padrão módulo 11 para verificar se as duas últimas posições da string passada, correspondem a dígitos verificadores válidos. Calculando primeiro o dígito das 12 primeiras posições e agregando o dígito encontrado ao fim da string, calcula o dígito das 13 posições, obtendo o segundo dígito. Retorna uma expressão lógica verdadeira se as duas últimas posições do CGC digitado coincidem com o calculado.

Sintaxe

CGC(ExpC1)

Parâmetros

ExpC1 – String de caracteres representando o número do C.G.C. sem pontos e traços separadores. Caso este argumento não seja passado para a função, esta considerará o GET corrente.

Retorna

ExpL1 – Expressão lógica .T. se o CGC é válido (dígito verificador confere) ou .F. se o dígito verificador não confere.

Exemplos:

cCGC:= Space(14):

@10,16 GET cCGC Picture "@R 99.999.999/9999-99" Valid CGC(cCGC)

A máscara do CGC deve vir com @R, para não inserir os pontos e a barra no CGC, o que impossibilita a validação.

@...CheckBox...Var

Tipo: Tela Windows

Cria uma caixa de verificação para definir entre Sim/Não ou Falso/Verdadeiro.

Sintaxe

@ nLinha,nColuna CHECKBOX cDesc VAR lSeleção Object oCbx

Parâmetros

nLinha – Número da linha superior
nColuna – Número da coluna superior
cDesc – Descrição da caixa. Ex. "Atualiza Estoque ?"
lSeleção – Variável Lógica que identifica se a caixa foi ou não selecionada
oCbx – Objeto associado ao Checkbox

Retorno

A variável <lSeleção> recebe ".T." se for selecionada ou ".F.", se vazia.

Comentários

Pode ser utilizada uma seqüência de CHECKBOX para determinar um conjunto de configurações onde vários itens podem ser marcados/desmarcados. Deve ser definida uma variável <lSeleção> para cada CHECKBOX definida.

Exemplo:

"Atualiza Estoque" - .T./.F. = Marcada/Desmarcada
"Gera Duplicata" - .T./.F. = Marcada/Desmarcada
"Calcula IPI" - .T./.F. = Marcada/Desmarcada

Anotações

CTOD()

Converte uma cadeia de caracteres em uma data correspondente

Sintaxe

CTOD(<cData>)

Argumentos

<cData> é uma cadeia de caracteres que contém números representando o mês, dia, e ano separados por qualquer outro caractere que não um número. Os dígitos do mês, dia, e ano devem ser especificados de acordo com o formato indicado pelo SET DATE. Para inicializar uma data vazia para uma entrada de dados, especifique a <cData> como sendo uma cadeia de caracteres nula (“ “).

ChkFile

Tipo: Processamento

Abre um arquivo do sistema, em modo exclusivo ou compartilhado, verificando a sua existência bem como dos índices, criando-os caso não existam. Esta função retorna verdadeiro (.T.) se o arquivo já estiver aberto ou se o Alias não for informado. Sempre que desejar mudar o modo de acesso do arquivo (de compartilhado para exclusivo ou vice-versa), feche-o antes de chamá-la.

Sintaxe

ChkFile(cAlias,lExcl,newAlias)

Parâmetros

cAlias – Alias do arquivo a ser aberto.

lExcl – Se for informado verdadeiro (.T.), o arquivo será aberto em modo exclusivo, caso contrário, o arquivo será aberto em modo compar-tilhado.

Se este parâmetro não for informado, será assumido falso (.F.).

newAlis – Abre o arquivo com outro apelido.

Retorna

lRet – Retorna verdadeiro (.T.) caso tenha conseguido abrir o arquivo e falso (.F.) caso contrário.

Exemplo

```
// Exemplo de uso da função ChkFile:
// Tenta abrir o arquivo de clientes como exclusivo:
dbSelectArea(“SA1”)
dbCloseArea()
lOk := .T.
While .T.
IF !ChkFile(“SA1”,.T.)
nResp := Alert(“Outro usuario usando! Tenta de novo?”,{“Sim”,”Nao”})
If nResp == 2
lOk := .F.
Exit
Endif
Endif
EndDo
```

```
If lOk
// Faz o processamento com o arquivo...
Endif
// Finaliza
If Select("SA1")
dbCloseArea()
Endif
ChkFile("SA1",.F.)
Return
```

Close

Tipo: Tela Windows

Desativa uma janela previamente definida e ativa.

Sintaxe

Close(cVar)

Parâmetros

cVar – Variável criada durante o comando de definição da janela.

Exemplo

```
@ 75,158 BmpButton type 02 Action Close(oDlg)
```

Anotações

CloseOpen

Tipo: Processamento

Função usada para fechar e abrir uma lista de arquivos.

Sintaxe

CloseOpen(aFecha,aAbre)

Parâmetros

aFecha – Array com a lista dos Aliases a serem fechados.
aAbre – Array com a lista dos Aliases a serem abertos.

Retorna

lRet – Retorna falso (.F.) se não conseguir abrir algum arquivo (se o arquivo estiver em uso exclusivo, por exemplo). Caso contrário, retorna verdadeiro (.T.).

Exemplo

```
// Exemplo de uso da funcao CloseOpen:
aFecha := {"SA1","SA2","SA3","SB1"}
aAbre := {"SG1","SH8"}
If CloseOpen(aFecha,aAbre)
.. Processamento
Endif
Return
```

Anotações

ClosesFile

Tipo: Processamento

Esta função fecha todos os arquivos, exceto os SXs, o SM2 e o SM4. Permite que se indique também outros arquivos que não devem ser fechados.

Sintaxe

ClosesFile(cAlias)

Parâmetros

cAlias – String com os Aliases dos arquivos que não devem ser fechados.
Devem ser informados separados por barras ("/")

Retorna

lRet – Retorna Verdadeiro (.T.) se fechou os arquivos com sucesso.
Retorna Falso (.F.), caso contrário.

Exemplo

```
// Exemplo de uso da funcao CLOSESFILE:
// Fecha todos os arquivos menos os cadastros:
```

```
cEmp := SM0->M0_CODIGO
ClosesFile("SA1/SA2/SA3/SA4/SA5/SA6/SA7/SA9/SAA/SAB/SAC")
// Processamento…
// Finalizacao
dbCloseAll()
OpenFile(cEmp)
Return
```

@...ComboBox...Itens...Size

Tipo: Tela Windows

Esta função é semelhante a LISTBOX, mas pode ser utilizada em pequenos espaços, pois os itens só serão mostrados quando a caixa for seleciona.

Sintaxe

@ nLinha,nColuna COMBOBOX cCont ITENS aArray SIZE nAltura,nLargura Object oCbx

Parâmetros

nLinha – Número da linha superior
nColuna – Número da coluna superior
cCont – Conteúdo caracter do item selecionado na Matriz [1]
aArray – Array, Matriz [1] com os itens para seleção
nAltura – Altura para definir o tamanho da caixa
nLargura – Largura para definir o tamanho da caixa
oCbx – Objeto relacionado ao botão

Retorno

O item selecionado pode ser obtido por <cCont>

Comentários

Os itens da Matriz [1] devem ser tipo "C" caracter.

Exemplo

Ver exemplo no programa RDDEMO apresentado no final deste Manual.

Anotações

ConfirmSX8

Tipo: Processamento

Permite a confirmação do número sugerido pelo Arquivo de Semáforo, através da função GETSX8NUM. Verifique a função GETSX8NUM para maiores detalhes.

Sintaxe

ConfirmSx8()

Exemplo

```
cNumSC5:=GetSX8Num("SC5")
Replace C5_NUM with cNumSC5
ConfirmSX8()
```

Verifique os exemplos descritos na função GETSX8NUM.

Anotações

Contar

Tipo: Processamento

Conta o número de registros de acordo com a condição determinada.

Sintaxe

Contar(cAlias, cCond)

Parâmetros

cALias – Alias do arquivo
cCond – Condição para a contagem

Exemplo

```
Contar("SC1",""C1_DATPRF < dDataBase")
```

CriaTrab

Tipo: Processamento

Cria arquivo de trabalho.

Sintaxe

CriaTrab(aArray,lDbf)

Parâmetros

aArray – Array multidimensional contendo os campos a criar {Nome, Tipo, Tamanho, Decimal}
lDbf – Determina se o arquivo de trabalho deve ser criado (.T.) ou não (.F.)

Retorna

ExpC1 – Nome do Arquivo gerado pela função.

Comentários

Esta função retorna o nome de um arquivo de trabalho que ainda não exista.

Caso lDbf = .T., a função criará um arquivo DBF com este nome e a estrutura definida em aArray.
Caso lDbf = .F., a função não criará arquivo de nenhum tipo, apenas fornecerá um nome válido.

Exemplos

```
// Com lDbf = .F.
cArq := CriaTrab(NIL, .F.)
cIndice := "C9_AGREG+"+IndexKey()
Index on &cIndice To &cArq
// Com lDbf = .T.
aStru := {}
AADD(aStru,{ "MARK" , "C", 1, 0})
AADD(aStru,{ "AGLUT" , "C", 10, 0})
AADD(aStru,{ "NUMOP" , "C", 10, 0})
AADD(aStru,{ "PRODUTO","C", 15, 0})
AADD(aStru,{ "QUANT" , "N", 16, 4})
AADD(aStru,{ "ENTREGA","D", 8, 0})
AADD(aStru,{ "ENTRAJU","D", 8, 0})
AADD(aStru,{ "ORDEM" , "N", 4, 0})
AADD(aStru,{ "GERADO" , "C", 1, 0})
cArqTrab := CriaTrab(aStru, .T.)
USE &cArqTrab ALIAS TRB NEW
```

CriaVar

Tipo: Processamento

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

Sintaxe

CriaVar(cCampo,lIniPad,cLado)

Parametros

cCampo – Nome do campo
lIniPad – Indica se considera (.T.) ou não (.F.) o inicializador
cLado – Se a variável for caracter, cLado pode ser: "C" - centralizado, "L"esquerdo ou "R" - direito
Retorna

uRet – Retorno (tipo de acordo com o dicionário de dados, considerando inicializador padrão)

Exemplo

```
// Exemplo do uso da função CriaVar:
cNumNota := CriaVar("F2_DOC") // Retorna o conteúdo do
// inicializador padrão,
// se existir, ou espaços em branco
Alert(cNumNota)
Return
```

Anotações

DTOS()

Converte um valor data para uma cadeia de caracteres com formato aaaammdd

Sintaxe

DTOS(<dData>) --> cData

Argumentos

<dData>, o valor data que ser convertido.

Retorna

DTOS() retorna uma cadeia com oito caracteres no formato, aaaammdd. Quando <dData> for uma data

nula (CTOD("")), DTOS() retorna uma cadeia de oito caracteres em branco. O valor de retorno nao, afetado pelo formato de data corrente.

Descrição

DTOS(), uma função de conversão de datas utilizada para criar expressoes de índice que consistem em um valor data e uma expressao caractere. DTOS() converte um valor data para uma cadeia de caracteres que pode ser concatenada a qualquer outra expressao caractere. O valor de retorno, estruturado para preservar a ordem de data (ano, mês, e dia).

Exemplos

Os exemplos a seguir ilustram DTOS() em conjunto com várias outras funções:

```
? DATE()                    // Resulta: 09/01/90
? DTOS(DATE())              // Resulta: 19900901
? LEN(DTOS(CTOD("")))       // Resulta: 8
```

Anotações

DATE()

Retorna a data do sistema como sendo um valor do tipo data

Sintaxe

DATE() --> dSistema

Retorno

DATE() retorna a data do sistema como sendo um valor do tipo data.

Descrição

DATE(), uma função de tratamento de datas que provê um meio de inicializar variáveis de memória com a data corrente, comparando outros valores do tipo data para a data corrente, e realizando operações aritméticas relativas … data corrente.

O formato para a exibição de datas, controlado pelo comando SET DATE.
O formato padrao assumido, mm/dd/aa.

Exemplos

Os exemplos seguintes mostram a função DATE() utilizada de varias maneiras:

```
? DATE()                 // Resulta: 09/01/90
? DATE() + 30            // Resulta: 10/01/90
? DATE() - 30            // Resulta: 08/02/90
dDate = DATE()
? CMONTH(dDate)      // Resulta: Setembro
```

Anotações

DataValida

Tipo: Processamento

Retorna uma data válida que não seja sábado, domingo ou feriado, a partir de uma data qualquer informada. É uma função útil para a geração de vencimentos reais para títulos, por exemplo.

Sintaxe

DataValida(dData)

Parametros

dData – Data informada para validação.

Retorna

dDtVld – Retorna a Data validada.

Exemplo

```
// Exemplo de uso da funcao DataValida:
// Pode-se gravar o campo do vencimento real de um
// titulo a partir do vencimento informado.
dVencto := cTod("")
Get dVencto

Read

dVencRea := DataValida(dVencto)
Grava() // Funcao generica.
// Um uso interessante, e a obtencao do numero de dias
// uteis de determinado mes utilizando-se dessa funcao.
```

```
// A logica e simples:
nDUtil := 0
nMes := 05
nAno := 98
dDtIni := CTOD("01/"+StrZero(nMes,2)+"/"+StrZero(nAno,2)
dDtMov := dDtIni
While Month(dDtIni) == Month(dDtMov) .And. Year(dDtIni) == Year(dDtMov)
If DataValida(dDtMov) == dDtMov
nDUtil := nDUtil + 1
Endif
dDtMov := dDtMov + 1
EndDo
```

Anotações

@...To...Dialog

Tipo: Tela Windows

Define uma nova janela na área de trabalho.

Sintaxe

@ nLinha1,nColuna1 TO nLinha2,nColuna2 DIALOG cVar TITLE cTítulo

Parâmetros

nLinha1 – Número da linha superior
nColuna1 – Número da coluna superior
nLinha2 – Número da linha inferior
nColuna2 – Número da coluna inferior
cVar – Variável que recebera as definições da nova janela
cTítulo – Titulo da Janela

Comentários

Deve ser utilizada sem conjunto com o comando ACTIVATE DIALOG.

Exemplo

Ver exemplo no programa RDDEMO apresentado no final deste Manual.

ExistChav

Tipo: Processamento

Verifica se a chave já existe em determinado Alias. Função para uso em validações de campos-chave, para não permitir a duplicidade de registros.

Sintaxe

ExistChav(cAlias,cChave,nOrdem,cHelp)

Parametros

cAlias – Alias do arquivo no qual a consistência deve ser avaliada
cChave – Chave para a pesquisa. Opcional. Se não for informada, o conteúdo será automaticamente obtido do GET ativo
nOrdem – Ordem do índice para a pesquisa no Alias. Se não for especificado, será assumida a primeira ordem
cHelp – Opcional chave de help. Se não for informada, o help será o padrão do sistema ("JAGRAVADO")
Retorna
lRet – Retorna Verdadeiro (.T.) se a chave não existir (o que significa que pode ser usada para incluir um novo registro). Caso contrário, retorna Falso (.F.) e executa um help do sistema.

Exemplo

```
// Exemplo de uso da funcao ExistChav:
// Pode-se utiliza-la em uma validacao de usuario,
// definada atraves do Configurador:
// Campo -> B1_COD
// Validacao do Usuario -> ExistChav("SB1")
// Ou em um AdvPL:
While .T.
cEsp := Space(15)
@ 00,00 Say "Cadastro de Especialidades"
@10,00 Say "Espec.:" Get cEsp Pict "@!"
Read
If LastKey() == 27
Exit
Endif
If ExistChav("SZ1",cEsp,1,"ESPEXIST")
Loop
Endif
Grava() // Rotina generica
EndDo
Return
```

EVAL()

Avalia um bloco de código

Sintaxe

EVAL(<bBloco>, [<ListaArgBloco>]) --> UltValorBloco
Argumentos

<bBloco>, o bloco de código a ser avaliado.
<ListaArgBloco>, uma lista de argumentos a ser enviada ao bloco de código antes que ele seja avaliado.

Retorno

EVAL() retorna o valor da última expressao dentro do bloco. Um bloco de código pode retornar um valor de qualquer tipo.

EOF()

Determina se o final do arquivo foi atingido.

Sintaxe

EOF()

Retorna verdadeiro(.T.) Quando é feita uma tentativa de mover o ponteiro de resgistros para além do último registro lógico em um arquivo de banco de dados; do contrário, ela retorna (.F.). Caso não haja nenhum arquivo de banco de dados aberto na área de trabalho corrente, EOF() retrona falso(.F.). Se o arquivo de banco de dados corretne não possui registros, EOF() retorna (.T.).

Anotações

ExistCpo

Tipo: Processamento

Verifica se determinada chave existe no Alias especificado. Função utilizada em processamentos no qual o código informado deve existir em determinado cadastro em sua validação.

Sintaxe

ExistCpo(cAlias,cChave,nOrdem)

Parâmetros

cAlias – Alias do arquivo para a pesquisa
cChave – Chave a ser pesquisada (opcional). Se não for informada, o conteúdo é obtido automaticamente do GET em uso nOrdem – Número da ordem do Índice para Pesquisa (Opcional). Se não for informado, usa a ordem atual do Alias.

Retorna

lRet – Retorna Verdadeiro (.T.) se a chave existir no Alias especificado, caso contrário, retorna Falso (.F.) e executa um help padrão do sistema ("REGNOIS"). A ordem utilizada na pesquisa é a atualmente selecionada. Se não for informado, usa a ordem atual do álias.

Exemplo

```
// Exemplo de uso da funcao ExistCpo:
While .T.
cProd := Space(15)
@10,10 Get cProd
Read
If LastKey() == 27
Exit
Endif
If !ExistCpo("SB1",cProd)
Loop
Endif
Grava() // Funcao generica.
EndDo
Return
```

Anotações

ExistIni

Tipo: Processamento

Verifica se o campo possui inicializador padrão.

Sintaxe

ExistIni(cCampo)

Parâmetros

cCampo – Nome do campo para verificação.
Retorna

lEx – Retorna Verdadeiro (.V.) se o campo possui inicializador padrão, caso contrário, retorna falso (.F.).

Exemplo

```
// Exemplo de uso da função ExistIni:
// Se existir inicializador no campo B1_COD:
If ExistIni("B1_COD")
// Chama o inicializador:
cCod := CriaVar("B1_COD")
Endif
Return
```

Anotações

Extenso

Tipo: Processamento

Gera o extenso de um valor numérico. Esta função retorna um valor, dinheiro ou quantidade, por extenso. Usada para a impressão de cheques, valor de duplicatas etc.

Sintaxe

Extenso(nValor,lQtd,nMoeda)

Parametros

nValor – Valor a ser gerado o extenso.
lQtd – Verdadeiro (.T.) indica que o valor representa uma quantidade.
Falso (.F.) indica que o valor representa dinheiro. Se não for especificado, o default é falso (.F.).
nMoeda - Qual moeda do sistema deve ser o extenso.

Retorna

cValor – Retorna o valor por extenso.

Exemplo

// Exemplo de uso da funcao Extenso:

```
nValor := SF2->F2_VALFAT
// Gera o extenso do valor, formatando a variavel com
// 200 caracteres preenchendo os espacos em branco com *
cExtenso := PADR(Extenso(nValor),200,"*")
// Imprime o extenso em duas linhas (100 caracteres em cada):
For nLi := 20 To 21
@nLi,10 Say Substr(cExtenso,(nLi-20)*100+1,100)
Next nLi
Return
```

Anotações

EMPTY()

Determina se o resultado de uma expressao , vazio

Sintaxe

EMPTY(<exp>) --> lVazio

Argumentos

<exp> , uma expressao de qualquer tipo de dados.

Retorno

EMPTY() retorna verdadeiro (.T.) se a expressao resultar em um valor vazio; do contrário, ela retorna falso (.F.). Os critérios para determinar se um valor , considerado vazio dependem do tipo de dados de <exp> de acordo com as seguintes regras:

FOUND( )

Determina se a operação de pesquisa anterior foi bem sucedida.

Sintaxe:

FOUND() —> lSucesso Retorno FOUND() retorna verdadeiro (.T.) se o último comando de pesquisa foi bem sucedido; do contrário, ela retorna falso (.F.). FOUND() é uma função de tratamento de banco de dados utilizada para determinar se uma operação de pesquisa (isto é, FIND, LOCATE, CONTINUE, SEEK, SET RELATION) foi bem sucedida antes que o próximo passo no programa seja executado.

FOUND() —> ISucesso Retorno FOUND() retorna verdadeiro (.T.) se o último comando de pesquisa foi bem sucedido; do contrário, ela retorna falso (.F.). FOUND() é uma função de tratamento de banco de dados utilizada para determinar se uma operação de pesquisa (isto é, FIND, LOCATE, CONTINUE, SEEK, SET RELATION) foi bem sucedida antes que o próximo passo no programa seja executado.

FCOUNT()

Retorna a quantidade de campos no arquivo (.dbf) corrente.

Sintaxe

FCOUNT() --> nCampos

Retorno

FCOUNT() retorna a quantidade de campos no arquivo de banco de dados aberto na rea de trabalho corrente na forma de um valor num,rico inteiro. Caso nao haja nenhum arquivo de banco de dados aberto, FCOUNT() retorna zero.

Descrição

FCOUNT(), uma função de tratamento de banco de dados. Ela, útil em aplicações que contêm programas independentes de dados os quais podem operar em qualquer arquivo de banco de dados. Nestes incluem-se programas gerais para importar e exportar dados e programas de relatórios. Normalmente, utiliza-se FCOUNT() para estabelecer o limite superior de um laço FOR...NEXT ou DO WHILE que processa um único campo por vez.

Anotações

Formula

Tipo: Processamento

Interpreta uma fórmula cadastrada. Esta função interpreta uma fórmula previamente cadastrada no Arquivo SM4 por meio do ambiente Configurador e retorna o resultado com tipo de dado de acordo com a própria fórmula.

Sintaxe

Formula(cFormula)

Parâmetros

cFormula – Código da fórmula cadastrada no SM4.

Retorna

uRet – Retorno, com tipo de dado de acordo com a fórmula.

Exemplo

```
// Exemplo de uso da funcao formula:
// Formula cadastrada no SM4:
// Codigo: F01
// Regra :"Sao Paulo,"+StrZero(Day(dDataBase),2)+
// " de "+MesExtenso(dDataBase)+" de "+
// StrZero(Year(dDataBase),4)+"."
// Ao imprimir esta linha em um programa, por exemplo,
@ 00,00 Say Formula("F01")
// o resultado impresso sera algo como:
// Sao Paulo, 17 de dezembro de 1997.
// Formula cadastrada no SM4:
// Codigo: F02
// Regra : (GETMV("MV_JUROS")/100)+1
// Ao usar esta formula, pode-se dar um acrescimo em um
// valor de acordo com a taxa de juros cadastrada no parametro MV_JUROS:
nValor := nValor * Formula("F02")
```

Anotações

@... GET

Tipo: Tela DOS/Windows

Executa um Get, diferenciado pela cláusula <F3>.

Sintaxe

@ nLinha,nColuna GET cVar <PICTURE> cMáscara <VALID> cFunção <F3> cConsulta

Parâmetros

nLinha – Número da Linha em que o Get será posicionado
nColuna – Número da Coluna em que o Get será posicionado
cVar – Variável a ser editada
cMáscara – Define a máscara de edição (opcional)
cFunção – Função que retorna valor lógico para validação da edição (opcional)
cConsulta – Definição (SXB) da consulta <F3> associada ao conteúdo de cVar
Comentários

Os códigos utilizados na cláusula <F3> devem ser obtidos por meio do arquivo (SXB). Não é necessário utilizar o comando READ após a definição dos Gets.

GetAdvFval

Tipo: Processamento

Esta função permite executar uma pesquisa em um arquivo pela chave e ordem especificadas, retornando o conteúdo de um ou mais campos.

Sintaxe

GetAdvFVal(cAlias,uCpo,uChave,nOrder,uDef)

Parâmetros

cAlias – Alias do arquivo.
uCpo – Nome de um campo ou array contendo os nomes dos campos
desejados.
uChave – Chave para a pesquisa.
nOrder – Ordem do indice para a pesquisa.
uDef – Valor ou array "default" para ser retornado caso a chave não seja encontrada.

Retorna

uRet – Retorna o conteúdo de um campo ou array com o conteúdo de vários campos.

Exemplo

```
// Exemplo de uso da funcao GetAdvFVal:

// Obtendo apenas de um campo:

cChave := SD2->D2_COD+SD2->D2_LOCAL
cDesc := GetAdvFVal("SB1","B1_DESC",cChave,1,SC6->C6_DESCRI)

// Obtendo o conteudo de mais de um campo:

cChave := SD2->D2_COD+SD2->D2_LOCAL
aCpos := {"B1_DESC","B1_PRV1","B1_UM"}
aDados := GetAdvFVal("SB1",aCpos,cChave,1,{SC6->C6_DESCRI,SC6->C6_PRCVEN,SC6->C6_UM})
```

refere-se aos Itens do Pedido de Venda) e, após pesquisar no SB1 (Cadastro de Produtos), sugerir a quantidade vendida a partir de um campo específico:

```
// Colunas...
nPosCod := aScan(aHeader,{ |x| Upper(AllTrim(x[2])) == "C6_PRODUTO" })
nPosQtd := aScan(aHeader,{ |x| Upper(AllTrim(x[2])) == "C6_QTDVEN" })

// Obtém o código do produto
```

```
cCodigo := aCols[n,nPosCod]

// Pesquisa

dbSelectArea("SB1")
dbSetOrder(1)
dbSeek(xFilial("SB1")+cCod)

// Altera a quantidade no grid

aCols[n,nPosQtd] := SB1->B1_QTSUGER // Campo específico com a quantidade padrão
__Return(SB1->B1_QTSUGER)
```

Para uma melhor compreensão, você pode analisar os programas RDMOD2.PRX e/ou RDMOD3.PRX que acompanham o SIGA MP8. Eles estão no diretório principal do sistema (geralmente \SIGAADV\) e demonstram rotinas usadas para cadastros semelhantes ao Pedido de Vendas e que trabalham com os arrays mencionados.

Anotações

GetMV

Tipo: Processamento
Recupera o conteúdo de parâmetros originados em SX6.

Sintaxe

GetMV(cParam)

Parâmetros

cParam – Nome do parâmetro a ser pesquisado no SX6

Retorna

ExpX1 – Conteúdo do parâmetro devolvido pela função

Exemplos

```
cTabVista := GETMV("MV_TABVIST")
cColICMS := GETMV("MV_COLICMS")
```

GetSX8Num

Tipo: Processamento

Fornece um número seqüencial do arquivo de semáforo (SX8??0.DBF).

Esta função retorna o próximo número, na seqüência e disponível, para o cadastro no SIGA MP8 e mantém esta numeração para o usuário até o momento em que ele confirme ou abandone a operação.

O arquivo de semáforo é usado para evitar a duplicidade de chaves em ambientes multiusuário. Esta função trabalha juntamente com outras duas chamadas CONFIRMSX8 e ROLLBACKSX8.

Verifique os exemplos para maiores detalhes.

Sintaxe

GetSx8Num(cAlias,cCpoSx8)

Parâmetros

cAlias – Alias do Arquivo
cCpoSx8 – Nome do campo para aplicação do semáforo

Exemplo

Para que o cadastro de clientes, por exemplo, carregue na inclusão o próximo número disponível automaticamente, pode-se utilizar a seguinte sintaxe no inicializador padrão do campo "A1_COD":
GetSx8Num("SA1")

Caso seja um arquivo específico, utilize a sintaxe a seguir:

GetSx8Num("SZ1","Z1_COD")

Para uso em programas AdvPL, as sintaxes descritas acima também são válidas, não devendo esquecer de que a função GETSX8NUM trabalha junto com as funções CONFIRMSX8 e ROLLBACKSX8, que devem ser chamadas ao final do processamento (procedimento que é feito automaticamente em um inicializador padrão conforme a sintaxe explicada acima).

Exemplo em AdvPL:

```
cCodNew := GetSx8Num("SZ1","Z1_COD")

// Processamento...

// Confirmacao

ConfirmSx8()

// ou Cancelamento

RollBackSx8()
Return
```

GetArea()

Descrição

Grava Caracteres da área atual (GetArea) em um array.

Sintaxe

ExpA1 := GetArea()

Parâmetros

ExpA1 = Expressão Array 1

Veja Também:

RestArea()
Tópicos Relacionados:
Exemplo de uso da Função GetArea
ExpA1 = GetArea()
RestArea(ExpA1)

Anotações

Help

Tipo: Tela DOS/Windows

Esta função exibe a ajuda especificada para o campo e permite sua edição. Se for um help novo, escreve-se o texto em tempo de execução.

Sintaxe

Help(cHelp,nLinha,cTítulo,cNil,cMensagem,nLinMen,nColMen)

Parâmetros

cHelp – Nome da Rotina chamadora do help (sempre branco)
nLinha – Número da linha da rotina chamadora (sempre 1)
cTítulo – Título do help
cNil – Sempre NIL
cMensagem – Mensagem adicional ao help
nLinMen – Número de linhas da Mensagem (relativa à janela)
nColMen – Número de colunas da Mensagem (relativa à janela)

Retorna
Nada

Exemplos

```
If Empty(cArqs)
dbSelectArea(cAlias)
RecLock(cAlias,.F.)
dbDelete()
Else
Help(" ",1,"NaoExclui",,cArqs,4,1)
Endif
```

Anotações

ImpCadast

Tipo: Impressão

Imprime relatório de cadastros padrões do SIGA MP8. Esta função monta uma interface padrão de relatório com parametrizações de/até e permite imprimir qualquer arquivo de cadastro definido no sistema.

Sintaxe

ImpCadast(cCab1,cCab2,cCab3,cNomePrg,cTam,nLim,cAlias)

Parâmetros

cCab1 – Primeira linha de cabeçalho
cCab2 – Segunda linha de cabeçalho
cCab3 – Terceira linha de cabeçalho
cNomePrg– Nome do programa
cTam – Tamanho do relatório ("P","M" ou "G")
nLim – Limite do relatório. Máxima coluna a ser impressa
cAlias – Alias do arquivo de cadastro a ser impresso

Exemplo

```
// Exemplo de uso da funcao Impcadast:
// Imprime relatorio de cadastro de produtos:
ImpCadast(Cabec1,Cabec2,Cabec3,"PRG01","P",80,"SB1")
Return
```

IncRegua

Tipo: Impressão

Incrementa régua padrão de processamento em relatórios.

Sintaxe

IncRegua()

Parâmetros

Nil

Retorno

Nil

Exemplo

```
DbSelectArea("SA1")
SetRegua(LastRec())
While ( ! Eof() )
@ Li, 001 PSAY SA1->A1_NOME
DbSkip()
IncRegua()
End
Comentário
Ver também SetRegua()
```

Anotações

IncProc

Tipo: Tela DOS/Windows

Incrementa régua padrão de processamento.

Sintaxe

IncProc()

Parâmetros

Nil

Retorno

Nil

Exemplo

```
ProcRegua(1000)
For i:= 1 to 1000
IncProc()
Next
Return
```

Anotações

INDEXORD()

Retorna a posição de ordem do índice controlador

Sintaxe

INDEXORD() --> nOrdem

Retorno

INDEXORD() retorna um valor num,rico inteiro. O valor retornado, igual … posição do índice controlador na lista de índices abertos na area de trabalho corrente. Um valor de zero indica que nao há índice controlador e que os registros estao sendo acessados em ordem natural.

Descrição

INDEXORD() , uma função de tratamento de banco de dados, que pode ser utilizada para determinar a posição do índice controlador na lista de arquivos de índices abertos pelo último comando USE...INDEX ou SET INDEX TO na área de trabalho corrente. Geralmente, útil para gravar o último índice controlador para que ele possa ser recuperado depois.

IndRegua

Tipo: Processamento

Cria índice de trabalho, podendo ser condicional.

Sintaxe

IndRegua(cAlias,cArqtrab,cChave,cPar,cFiltro,cTexto)

Parâmetros

cAlias – Alias do arquivo.
cArqtrab – Nome do arquivo de trabalho retornado pela função CriaTrab (.F.).
cChave – Expressão utilizada na chave do novo índice.
cPar – Se for igual a 'D', cria um indice com a chave inversa, do maior valor para o menor.
cFiltro – Expressão utilizada para filtro.
cTexto – Texto da régua de processamento ("Selecionando registros ..."). RetornoNil

Exemplo

```
DbSelectArea("SC5")
cFiltro := "C5_OK<>'X'"
cChave := "Dtos(C5_EMISSAO)+C5_VEND1"
cIndSC51 := CriaTrab(Nil,.F.)
IndRegua("SC5",cIndSC51, cChave, , cFiltro,"Selecionando Pedidos...")
```

Anotações

LEN()

Retorna o tamanho de um string ou a quantidade de elementos em um vetor

Sintaxe

LEN(<cString> | <aDestino>) --> nCont

Argumentos

<cString>, a cadeia de caracteres cujo tamanho ser medido.
<aDestino>, o vetor cujos elementos serao contados.

Retorno

LEN() retorna o tamanho de uma cadeia de caracteres ou a quantidade de elementos em um vetor na forma de um valor num,rico inteiro. Caso a cadeia de caracteres seja nula ("") ou o vetor esteja vazio, LEN() retorna zero.

LetterOrNum

Tipo: Processamento

Verifica se determinado caractere é uma letra ou um número.

Sintaxe

LetterOrNum(cChar)

Parâmetros

cChar – Caracter para verificação.

Retorna

lAlfa – Retorna Verdadeiro (.V.) se o caracter informado for uma letra ou um número.

Exemplo

```
// Exemplo de uso da funcao LetterOrNum:

cCh := Inkey(0)
If LetterOrNum(cCh)
... Processamento
Endif
Return
```

Anotações

MarkBrowse

Tipo: Processamento

Monta um browse padrão do sistema, permitindo marcar e desmacar linhas.

Sintaxe

MarkBrowse(cAlias,cCampo,cCpo,aCampos,lMarc,cMarca,cCtrlM,lBotoes,cTopFun,cBotFun,aCoord)

Parâmetros

cAlias – Alias do arquivo
cCampo – Campo que estabelece relação com a culuna de marca
cCpo – Campo que se estiver vazio muda a cor da linha
aCampos – Array com os campos para montar o browse
lMarc – Flag para inicializar marcado ou não

cMarca – Marca obtida com a função Getmark
cCtrlM – Função para ser executada no Alt_M
lBotoes – Parâmetro obsoleto
cTopFun – Função filtro para estabelecer o primeiro registro
cTopFun – Função filtro para estabelecer o último registro
aCoord – Array com as coordenadas da MarkBrowse.

Exemplo

```
cMarca := GetMark()
cCadastro := "Escolher Clientes"
aRotina := { { "Pesquisar","AxPesqui",0,1}, ;
{"Visualizar","AxVisual",0,2}}
MarkBrow("SA1","A1_OK","SA1->A1_OK",,,cMarca)
```

Anotações

MBrowse

Tipo: Processamento

Monta um browse padrão do sistema, conforme os parâmetros.

Sintaxe

mBrowse(nLinha1, nColuna1, nLinha2, nColuna2, cAlias, aFixe, cCpo, nPar, cCor, n Opc)

Parâmetros

nLinha1 – Número da linha inicial
nColuna1 – Número da coluna inicial
nLinha2 – Número da linha final
nColuna2 – Número da coluna final
cAlias – Alias do arquivo
aFixe – Array contendo os campos fixos (a serem mostrados em primeiro lugar no browse)
cCpo – Campo a ser tratado. Quando vazio, muda a cor da linha
nPar – Parâmetro obsoleto
cCor – Função que retorna um valor lógico, muda a cor da linha
nOpc – Define qual opção do aRotina que será utilizada no double click
MBrowse(6, 1, 22, 75, "SA1")

MesExtenso

Tipo: Processamento

Retorna o nome do mês por extenso.

Sintaxe

MesExtenso(nMes)

Parâmetros

nMes – Número do mês (1 a 12). Se "nMes" não for informado, é assumido o mês da data base do sistema. Esta variável também pode ser caracter ("1" ou "2") ou do tipo data.

Retorna

cNome – Nome do mês retornado por extenso.

Exemplo

```
// Exemplo do uso da funcao MesExtenso:

? "Sao Paulo,"+STRZERO(Day(dDataBase),2)+" de "+;
MesExtenso(dDataBase)+" de "+StrZero(Year(dDataBase),4)
```

Anotações

Modelo2

Tipo: Processamento

Exibe formulário para cadastro segundo o modelo 2 (como a rotina de Nota Fiscal).

Sintaxe

Modelo 2 (cTítulo ,a Cabec ,a R o d a p é ,a G d ,n O p ,c L O k ,c T O k , [aGetsGD,bF4,cIniCpos,nMax,aCord w,lDelget])

Parâmetros

cTítulo – Título da janela
aCabec – Array com os campos do cabeçalho

aRodapé – Array com os campos do rodapé
aGd – Array com as posições para edição dos itens (GETDADOS)
nOp – Modo de operação (3 ou 4 altera e inclui itens, 6 altera mas não
inclui itens, qualquer outro número só visualiza os itens)
cLOk – Função para validação da linha
cTOk – Função para validação de todos os dados (na confirmação)
aGetsGD – Array Gets editáveis (GetDados)
Default = Todos.
bF4 – Codeblock a ser atribuído a tecla F4.
Default = Nenhum.
cIniCpos – String com o nome dos campos que devem ser inicializados ao teclar seta para baixo (GetDados).
nMAx – Limita o número de linhas (GetDados).
Default = 99.
aCordw – Array com quatro elementos numéricos, correspondendo às coor-denadas
linha superior, coluna esquerda, linha interior e coluna
direita, definindo a área de tela a ser usada.

Default = Área de Dados Livre.

lDelget – Determina se as linhas podem ser deletadas ou não (GetDados)
Default = .T.

Retorna

lRet – Retorna .T. se for confirmado
lRet:=Modelo2(cTitulo,aC,aR,aCGD,nOpcx,cLinhaOk,cTudoOk)

Anotações

Modelo3

Sintaxe

Modelo3(cTitulo,cAliasEnchoice,cAliasGetD,aCpoEnchoice,cLinOk,cTudOk,nOpcE,nOpcG,cFieldOk, [lVirtual,nLinhas,aAltEnchoice])

Parâmetros

cTitulo – Título da janela
cAliasEnchoice – Álias do cabeçalho
cAliasGetd – Álias dos itens
aCpoEnchoice – Array com os campos que serão mostrados
cLinOk – Função para validar a mudança de linha nos itens.

cTudOk – Função para validar todos os itens.
nOpce – Número da opção do menu para o cabeçalho (Enchoice)
nOpcG – Número da opção do menu para o itens (GetDados)
cFieldOk – Função para validar os campos dos itens (GetDados)
lVirtual – Permite visualizar campos virtuais na enchoice.
Default = .F.
nLinhas – Limita o número máximo de linhas (GetDados)
Default = 99.
aAltEnchoice – Array com campos alteráveis (Enchoice)
Default = Todos.

Retorna

lRet – Retorno da função modelo3. Se True a operação foi confirmada.

_lRet:=Modelo3(cTitulo,cAliasEnchoice,cAliasGetD,aCpoEnchoice,cLinOk,cTudOk,nOpcE,nOpcG,cFieldOk)

Anotações

MontaF3

Tipo: Processamento

Permite o acesso à janela de consultas padronizadas (criadas no SXB) por meio de um GET usando a tecla F3.

Sintaxe

MontaF3(cAlias)

Parâmetros

cAlias – Alias do arquivo ou código de tabela para consulta. Se não informado, desabilita a tecla F3.

Exemplo

// Exemplo de uso da funcao MontaF3:

// Versao DOS

cCod := Space(15)

```
@02,50 Say "Digite o codigo:" Get cCod Picture "@!";
When MontaF3("SB1") Valid(MontaF3())
Read
Return

// ***************************
// Versao WINDOWS
// Use a propria clausula do comando get:
@ 250,250 To 360,450 Dialog oDlg Title "Teste"
@ 02,50 Get cCod Picture "@!" F3 "SB1"

Activate Dialog oDlg Centered

Return
```

Anotações

MsgBox

Tipo: Tela Windows

Abre uma caixa de diálogo padronizada para informar ao usuário de um erro, decisão a ser tomada ou apenas uma informação ("Registro Gravado com sucesso").

Sintaxe

MSGBOX(cMensagem,cTítulo,cTpCaixa)

Parâmetros

cMensagem – Define a mensagem apresentada no interior da janela
cTítulo – Titulo da janela
cTpCaixa – Tipo da caixa padronizada

Retorno

Retorna Nil ou um valor lógico (.T. ou .F.) conforme o tipo de caixa.

Comentários

As caixas assumem o tamanho de <cMensagem>.

Tipos de caixas:

"STOP", utiliza um bitmap para advertência e tem um botão "Ok". Retorna Nil.
"INFO", utiliza um bitmap para advertência e tem um botão "Ok". Retorna Nil.
"ALERT", utiliza um bitmap para advertência e tem um botão "Ok". Retorna Nil.
"YESNO", utiliza um bitmap para advertência e tem dois botões "Sim" e "Não", retorna .T. ou .F.
"RETRYCANCEL", utiliza um bitmap para advertência e tem dois botões "Repetir" e "Cancelar", retorna .T. ou .F.

MsUnLock()

Libera lock de registro.

Sintaxe

MsUnLock()

Descrição

A função MsUnlock() libera os registros bloqueados pela função RecLock().

Não retorna valores.

Exemplo

```
RecLock( 'XXX' ,.F. )
Replace Campo With '000001'
MsUnLock()
```

@..To...MultiLine

Tipo: Tela Windows

Ativa Browse para edição de múltiplos itens padrão SigaAdv Win (GetDados Itens SC6).

Sintaxe

@ nLinha1,nColuna1 TO nLinha2,nColuna2 MULTILINE <<MODIFY>> <<DELETE>> <<VALID>> cFunção <<FREEZE>> nColuna

Parâmetros

nLinha1 – Número da linha superior
nColuna1 – Número da coluna superior
nLinha2 – Número da linha inferior

nColuna2 – Número da coluna inferior
cFunção – Função a ser executada para validar mudança de linha <opcional>
nColuna – Número de colunas "Congeladas à esquerda" <opcional>

Comentários

As cláusulas opcionais permitidas controlam as alterações, exclusões e validações nas mudanças de linha e congelamento de colunas, respectivamente.

Devem ser criadas obrigatoriamente as matrizes aHeader [n][n] e aCols[n][n] antes da definição da MULTILINE, sendo que aHeader [n][n] contém informações sobre os campos que serão editados (SX3) e aCols [n][n] contém os dados referentes aos campos que serão editados.

Anotações

NaoVazio

Tipo: Processamento

Verifica se o campo não está vazio.

Sintaxe

NaoVazio(cCpo)

Parâmetros

cCpo – Campo a verificar

Retorna

ExpL1 – Valor Lógico de Retorno (.T. ou .F.)

Exemplos

@ 5,10 Get cCodigo Valid NaoVazio(cCodigo)

Ms_Flush()

Descarrega spool de impressão.

Sintaxe

Ms_Flush()

Descrição

Após os comandos de impressão as informações ficam armazenadas no spool e são descarrega em seus destinos através da função Ms_Flush().

Negativo

Tipo: Processamento

Verifica se é negativo.

Sintaxe

Negativo(nCpo)

Parâmetros

nCpo – Campo a verificar

Retorna

ExpL1 – Se o valor de nCpo for menor que 0, é retornado .T., caso contrário será retornado .F..

Exemplos

If Negativo (nValTitulo)

@ 5,10 Say "Valor invalido"

Loop
EndIf

OpenFile

Tipo: Processamento

É a função que exibe o diagnóstico de arquivos, verificando a existência dos arquivos de dados e dos índices do sistema, criando-os caso não existam, etc.

Abre os arquivos de acordo com o ambiente onde é executada ou de acordo com a parametrização.

Sintaxe

OpenFile(cEmp)

Parâmetros

cEmp – Código da empresa que deve ser aberta.

Exemplo

```
// Exemplo de uso da funcao openfile:

cEmp := SM0->M0_CODIGO

// Elimina os indices de todos os arquivos abertos no

// SX2 para reindexacao

dbSelectArea("SX2")
dbGoTop()
While !EOF()
If Select(SX2->X2_CHAVE) > 0
dbSelectArea(SX2->X2_CHAVE)
dbCloseArea()
cEsp := AllTrim(SX2->X2_PATH)
cEsp := cEsp + AllTrim(SX2->X2_ARQUIVO) + "*" + RetIndExt()
fErase(cEsp)
Endif
dbSelectArea("SX2")
dbSkip()
EndDo
dbCloseAll() // Fecha todos os arquivos
OpenFile(cEmp) // Abre os arquivos (reindexa)
Return
```

Parâmetro cEmp apenas no Windows.

OurSpool

Tipo: Impressão

Abre a tela do gerenciador de impressão do sistema. O gerenciador mostra os relatórios impressos em disco gravados no diretório definido por meio do parâmetro

"MV_RELATO".

Sintaxe

OurSpool(cArq)

Parâmetros

cArq – Nome do arquivo. Este parâmetro é opcional, se informado, o gerenciador de impressão já é aberto neste arquivo.

Exemplo

```
// Exemplo de uso da funcao ourspool:
// Ativa o gerenciador de impressao:

OurSpool()

// Para verificar o uso desta funcao em relatorios,
// verifique o exemplo da funcao AVALIMP.

Return
```

Anotações

Obrigatorio

Descrição

Verifica se todos os campos obrigatórios foram digitados.

Sintaxe

ExpL1 := Obrigatorio(aGets,aTela,tObg)

Parâmetros

aGets = Retorna .F. se algum campo obrigatório está vazio
aTela = Array com os campos a serem verificados
tObg = Array com os títulos da tela

Tópicos Relacionados:

Exemplo de uso da Função Obrigatorio

Pergunte

Tipo: Impressão

Esta função permite acessar e editar um grupo de perguntas do arquivo SX1. Mostra uma tela contendo as perguntas gravadas em SX1 a serem respondidas ou confirmadas pelo usuário.

Sintaxe

Pergunte(cGrupo, lVar)

Parâmetros

cGrupo – Nome do Grupo de perguntas.

lVar – .F. - devolve o conteúdo das variáveis, não apresentando a janela de perguntas; .T. - permite a alteração das variáveis, apresentando a janela.

Retorna

ExpL1 – .T. se o grupo de perguntas existe.

Exemplos

pergunte("AFR090",.T.)

```
// Variáveis utilizadas na pergunta
// mv_par01 a partir da data
// mv_par02 até a data
// mv_par03 da conta
// mv_par04 até a conta
// mv_par05 do centro de custo
// mv_par06 até o centro de custo
// mv_par07 do código
// mv_par08 até o código
// mv_par09 do projeto
// mv_par10 até o projeto
// mv_par11 moeda
```

Pertence

Tipo: Processamento

Verifica se o campo está contido em outro.

Sintaxe

Pertence(cString,cCampo)

Parâmetros

cString – String que deve estar contida no cCampo
cCampo – Campo a verificar
Retorna

ExpL1 – Valor Lógico de Retorno (.T. ou .F.)

Exemplo

```
SetCursor(1)

@ 09,19 Get cTipo Picture "@!" Valid Pertence("CL\VD\PD",cTipo)
```

PesqPict

Tipo: Processamento

Pesquisa, no dicionário de dados, qual a picture usada em um determinado campo, seja para a edição em tela ou para a impressão de relatórios.

Sintaxe

```
PesqPict(cAlias,cCampo,nTam)
```

Parâmetros

cAlias – Alias do arquivo
cCampo – Nome do campo
nTam – Opcional, para campos numéricos, será usado como o tamanho do campo para definição da picture. Se não informado, e usado o tamanho padrão no dicionário de dados.
Retorna
cPic – Picture do campo

Exemplo

```
// Exemplo de uso da funcao PesqPict
// Em um relatorio pode-se usar a sintaxe abaixo para
// formatacao automatica de acordo com o dicionario de
// dados:

@ nLin,20 Say "Total:"
@ nLin,27 Say SF2->F2_VALBRUT Picture PesqPict("SF2","F2_VALBRUT")
// ...
Return
```

PesqPictQt

Tipo: Processamento

Devolve a picture de um campo de quantidade, de acordo com o dicionário de dados. Esta função geralmente é utilizada quando há pouco espaço disponível para impressão de valores em relatórios, quando o valor nEdição não é informado.

Ela tem o comportamento semelhante ao da função "X3Picture", pois busca o tamanho do campo no dicionário de dados.

Sintaxe

PesqPictQt(cCampo,nEdição)

Parâmetros

cCampo – Nome do campo a verificar a picture
nEdição – Espaço disponível para edição
Retorna

ExpC – Picture ideal para o espaço definido por nEdição, sem a separação dos milhares por vírgula

Exemplo

@ 8,10 Say SB2->B2_QATU Picture PesqPictQt ("B2_QATU",8)

Anotações

Posicione

Tipo: Processamento

Posiciona o arquivo em um determinado registro.

Sintaxe

Posicione(cAlias, nOrdem, cChave, cCampo)

Parâmetros

cAlias – Alias do arquivo
nOrdem – Ordem utilizada
cChave – Chave pesquisa
cCampo – Campo a ser retornado
Retorna

Retorna o conteúdo do campo passado com o perímetro.

Exemplo

Posicione("SA1",1,xFilial("SA1")+001,"A1_NOME")

Positivo

Tipo: Processamento

Verifica se é positivo.

Sintaxe

Positivo(nCampo)

Parâmetros

nCampo – Campo a verificar

Retorna

ExpL1 – Valor Lógico de Retorno (.T. ou .F.)

Comentários

Se cCampo for maior ou igual (>=) a zero a função retorna .T.
Caso contrário retorna .F.

Exemplo

@ 09,07 Get nValor Picture "999999" Valid Positivo (nValor)

PswAdmin( )

Verifica se um usuário é Administrador.

Sintaxe

PswAdmin(cUsuario, cSenha)

Argumento	Obrigat.	Tipo	Descrição
cUsuario	Sim	C	Nome do usuário.
cSenha	Sim	C	Senha do usuário.

Descrição

Esta função retorna um valor lógico, se for usuário administrador verdadeiro (.T.), caso contrário falso (.F.).

Exemplo

```
User Function <nome-da-função>( cUsuario, cSenha )
Local lAdminst := .F.
If PswAdmin( cUsuario, cSenha )
lAdminst := .T.
Else
APMsgInfo( "Usuário não é Administrador !" )
EndIf
Return lAdminst
```

ProcRegua

Tipo: Tela DOS/Windows

Inicializa régua padrão de processamento.

Sintaxe

ProcRegua(nRegs,nLinha,nColuna)

Parâmetros

nRegs – Número de registros que serão processados.
nLinha – Número da Linha da régua
nColuna – Número da Coluna da régua

Retorna

Nil

Exemplo

```
ProcRegua(1000)
For i:= 1 to 1000
IncProc()
Next
Return
```

No programa para Windows a ProcRegua só utiliza o primeiro parâmetro. No programa para DOS são utilizados os três parâmetros.

= Ver também IncProc()

ProxReg

Tipo: Processamento

Retorna o último registro incrementado.

Esta função retorna um valor, numérico ou caracter, contendo o próximo número a partir do último registro encontrado.

O campo que é levado em consideração é aquele que se encontra posicionado no SX3 (dicionário de dados). Pode ser usada para obter os próximos valores para campos dos tipos: Caracter, Numérico e Data.

Sintaxe

ProxReg(nInc,nPos,nIndice)

Parâmetros

nInc – Valor a incrementar
nPos – Tamanho
nÍndice – Número do índice a ser utilizado
Retorna
uRet – Próximo número (último registro incrementado)

Exemplo

// Exemplo de uso da função ProxReg:

```
dbSelectArea("SX3")
dbSetOrder(2)
dbSeek("A1_COD")
dbSelectArea("SA1")
cProx := ProxReg(1,6,1) // Retorna o possível próximo

// código para o cadastro de
// cliente

dbSelectArea("SX3")
dbSeek("D2_NUMSEQ")
dbSelectArea("SD2")
nProx := ProxReg(1,,4) // Retorna o próoximo número

// seqüencial
```

Return
Processa()
Cria diálogo com uma régua de progressão.
Sintaxe
Processa(bAcao, [cTitulo] , [cMsg], [lAborta])

Argumento	Obrigat.	Tipo	Descrição
cAcao	Sim	Código de bloco	Função a ser executada.
cMsg	Não	C	Mensagem a ser exibida a baixo da régua de progressão
lAborta	Não	L	Habilita botão cancelar.
cTitulo	Não	C	Título de janela

Descrição

A função Processa() cria um diálogo onde a execução de um determinada função pode ser acompanhada atráves de uma régua de progressão. Para atribuir o valor total da régua utilizamos a função ProcRegua() e para incrementar a régua utilizamos a função IncProc().

Exemplo

```
User Function <nome-da-função>( )
Local bAcao := {|lFim| Exemplo(@lFim) }
Local cTitulo := ''
Local cMsg := 'Processando'
Local lAborta := .T.
Processa( bAcao, cTitulo, cMsg, lAborta )
Return
Static Function Exemplo(lFim)
Local nI
ProcRegua(10000)
For nI := 1 To 10000
If lFim
Exit
EndIf
IncProc()
Next nI
Return
```

Anotações

@...Radio

Tipo: Tela Windows

Cria uma caixa de seleção semelhante a CHECKBOX. Todos os itens são Apresentados, mas apenas um pode ser selecionado.

Sintaxe

@ nLinha,nColuna RADIO aArray VAR nPos Object oRdx

Parâmetros

nLinha – Número da linha superior
nColuna – Número da coluna superior
aArray – Matriz [1] com os Itens
nPos – Contém a posição na Matriz[1] do item selecionado
oRdx – Objeto associado à Radiobox()

Retorno

O item selecionado pode ser obtido por - "Matriz [n3]"

Comentários

Os itens da Matriz [1] devem ser do tipo "C" caracter. Pode ser utilizada para definir uma característica em um conjunto. Ex.Tipo da Condição de pagamento

· Tipo 1
· Tipo 2
· Tipo 3

RecLock

Tipo: Processamento

Tenta efetuar um lock no registro do banco de dados informado.

Sintaxe

RecLock(cAlias,lAdiciona)

Parâmetros

cAlias – Alias do Banco de Dados

lAdiciona – .T. adiciona registro ao Banco de Dados

Retorna

ExpL1 – Valor Lógico de Retorno (.T. ou .F.)

Comentários

Esta função tenta colocar o registro corrente do arquivo cAlias em lock.

É necessário colocar um registro em lock sempre que for efetuar uma atualização, como um comando Replace ou um Delete. Caso lAdiciona = .T., a função RecLock inclui (com um Append Blank) um registro no arquivo cAlias. Se a operação for bem sucedida, retorna .T.

Exemplo

```
// Exclusão de Registro

// Com lAdiciona = .F.

If ! RecLock("SF1",.F.)

@ 1,1 Say "Registro em uso por outra estação"

Loop
EndIf
dbDelete()
dbUnLock()
Com ExpL2 = .T.

// Inclusão de Registro

RecLock("SF1",.T.)
Replace F1_TIPO With cTipo, F1_DOC With cNFiscal,;
F1_SERIE With cSerie ,F1_EMISSAO With dDEmissao,;
F1_LOJA With cLoja ,F1_FORNECE With Subs(cA100For,1,6)
dbUnLock()
```

Após a atualização do registro, deve-se executar a função MsUnlock().

RollBackSX8

Descrição

Retorna o número obtido pela função GetSX8Num no semáforo como pendente. Verifique a função GetSX8Num para maiores detalhes.

Sintaxe

RollBackSx8()

Parâmetros

Veja Também:

GetSX8Num

Tópicos Relacionados:

Exemplo de uso da Função RollBackSX8

RestArea

Tipo: Processamento

Restaura a área RestArea a partir do array.

Sintaxe

RestArea(aArray)

Parâmetros

aArray – Expressão Array para restauração

Exemplo

aArray:=GetArea()
RestArea(aArray)
Ver também
Função GetArea( )

Anotações

RetASC

Tipo: Processamento

Retorna um código de letras quando ultrapassar o número máximo de dígitos.

Sintaxe

RetAsc(cOri,nTam,lAlfa)

Parâmetros

cOri – String original a ser convertida.
nTam – Tamanho máximo para a conversão.
lAlfa – Indica se o retorno deve conter letras (.T.) ou somente números (.F.)

Retorna

cStr – Retorna a String formada com letras ou números.

Exemplo

```
// Exemplo de uso da funcao RetAsc:

// Suponha a existencia de um campo caracter de tamanho

// 2. Usando a funcao RetAsc com o parametro lAlfa ver-
// dadeiro (.T.) se o numero ultrapassar 99 retornara A0
cCod := StrZero(ProxReg(1,2,1),2)
// Se ultrapassar 99 retorna A0
cCod := RetAsc(cCod,2,.T.)
__Return(cCod)
```

RetIndex

Tipo: Processamento

Devolve os índices padrões do SIGA.

Sintaxe

RetIndex(cAlias)

Parâmetros

cAlias – Alias do Arquivo

Retorna

Número Índices existentes no SINDEX

Comentários

Baseado no SINDEX, abre todos os índices padrões para o arquivo em pauta.

Exemplo

```
Select SA1
Index on A1_ACUM to TRAB
SINDEX := RetIndex ("SA1")
```

Anotações

RollBackSX8

Tipo: Processamento

Retorna o número obtido pela função GETSX8NUM no semáforo como pendente. Verifique a função GETSX8NUM para maiores detalhes.

Sintaxe

RollBackSx8()

Exemplo

Verifique os exemplos na função GETSX8NUM.

RECNO()

Retorna o n£mero do registro corrente de uma rea de trabalho

Sintaxe

RECNO() --> nRegistro

Retorno

RECNO() retorna o n£mero do registro corrente na forma de um valor num,rico inteiro. Se a rea de trabalho cont,m um arquivo de banco de dados com zero registros, RECNO() retorna um, BOF() e EOF() retornam verdadeiro (.T.), e LASTREC() retorna zero.

Se o ponteiro de registros for movido para al,m do £ltimo registro, RECNO() retorna LASTREC() + 1 e EOF() retorna verdadeiro (.T.). Caso seja feita uma tentativa para mover o ponteiro al,m do primeiro registro, RECNO() retorna o n£mero do primeiro registro lógico no arquivo de banco de dados, e BOF() retorna verdadeiro (.T.).

RptStatus

Tipo: Processamento (Apenas Windows)

Executa função de detalhe do relatório.

Sintaxe

RptStatus(bBlock) => RptStatus(bBlock, cTítulo, cMsg)

Parâmetros

bBlock – Bloco de código que define a função a ser executada.
cTítulo – Título do diálogo de processamento.
cMsg – Mensagem do diálogo de processamento.

Comentários

Pode ser utilizada com os parâmetros:

RptStatus({ | | Execute("Nome da Função") })

RegToMemory

Descrição

Cria variaveis M-> para uso na Modelo3()

Sintaxe

RegToMemory(cAlias,lInc)

Parâmetros

cAlias =
lInc =
Veja Também:

Anotações

Modelo3

Tópicos Relacionados:

Exemplo de uso da Função RegToMemory

SetDefault

Tipo: Processamento

Habilita os padrões definidos pela função SetPrint.

Sintaxe

SetDefault (aArray, cAlias)

Parâmetros

aArray – Array aReturn, preenchido pelo SetPrint
[1] Reservado para Formulário

[2] Reservado para Nº de Vias
[3] Destinatário
[4] Formato => 1-Comprimido 2-Normal
[5] Mídia => 1-Disco 2-Impressora
[6] Porta ou Arquivo 1-LPT1... 4-COM1...
[7] Expressão do Filtro
[8] Ordem a ser selecionada
[9]..[10]..[n] Campos a Processar (se houver)
cAlias – Alias do arquivo a ser impresso.

Retorna

Nil

Comentários

Esta função habilita os padrões de relatório alterados pela função SetPrint.

Exemplo

```
// Define Variáveis

cString:= "SB1"
NomeRel:= "MATR290"
cPerg := "MTR290"
titulo := "RELAÇÃO PARA ANÁLISE DOS ESTOQUES"
cDesc1 := "Este relatório demonstra a situação de cada item em "
cDesc2 := "relação ao seu saldo , seu empenho , suas entradas previstas"
cDesc3 := "e sua classe ABC."
aOrd := {" Por Codigo "," Por Tipo "}
Tamanho := "G"

// Envia controle para a função SETPRINT

NomeRel:= SetPrint( cString, NomeRel, cPerg, @titulo, cDesc1, ;
cDesc2, cDesc3, .F., aOrd, .F., Tamanho)
If LastKey() = 27 .or. nLastKey = 27
RestScreen(3,0,24,79,cSavScr1)
Return
Endif
SetDefault(aReturn,cAlias)
If LastKey() = 27 .OR. nLastKey = 27
RestScreen(3,0,24,79,cSavScr1)
Return
Endif
```

SetDlg

Tipo: Tela Windows

Colocar um título em uma Dialog.

Sintaxe

SetDlg(oWnd, cText)

Parâmetros

oWnd – Objeto da janela
cText – Novo Texto

Exemplo

```
If ( INCLUI )
cCaption := 'Inclusao de Pedidos'
ElseIf ( ALTERA )
cCaption := 'Alteracao de Pedidos'
EndIf
SetDlg( oDlg, cCaption )
```

SUBSTR()

Extrai um substring de uma cadeia de caracteres

Sintaxe

SUBSTR(<cString>, <nIn¡cio>, [<nCont>]) --> cSubstring

Argumentos

<cString> , a cadeia de caracteres da qual ser extraido um substring, podendo ter at, 65.535 (64K) bytes, que , o tamanho máximo de uma cadeia de caracteres em Clipper.

<nInicio> , a posição inicial em <cString>. Se <nIn¡cio> for positivo, ele, relativo ao caractere mais … esquerda em <cString>. Se <nIn¡cio> for negativo, ele, relativo ao caractere mais … direita em <cString>. <nCont> , a quantidade de caracteres a serem extraidos. Se omitido, o substring come‡a em <nIn¡cio> e continua at, o fim da cadeia de caracteres. Se <nCont> for maior do que a quantidade de caracteres existentes a partir de <nInício> at, o final de <cString>, o excedente , ignorado.

Retorno

SUBSTR() retorna uma cadeia de caracteres.

Descrição

SUBSTR() , uma função de tratamento de caracteres que extrai um substring de qualquer outra cadeia ou campo memo. SUBSTR() esta relacionada …s funções LEFT() e RIGHT(), que extraem substrings que começam com os caracteres mais … esquerda e mais … direita em <cString>, respectivamente.

As funções SUBSTR(), RIGHT(), e LEFT() sao utilizadas juntamente com as funções AT() e RAT() para localizar a primeira e/ou última posição de um substring antes de extrai-lo. Elas também sao utilizadas para exibir ou imprimir apenas uma parte de uma cadeia de caracteres.

Exemplos

Os exemplos a seguir extraem o primeiro e o £ltimo nome de uma variável:

```
cName := "Biff Styvesent"
? SUBSTR(cNome, 1, 4)                  // Resulta: Biff
? SUBSTR(cNome, 6)                     // Resulta: Styvesent
? SUBSTR(cNome, LEN(cNome) + 2)        // Resulta: null string
? SUBSTR(cNome, -9)                    // Resulta: Styvesent
? SUBSTR(cNome, -9, 3)                 // Resulta: Sty
```

SetPrint

Tipo: Impressão

Altera os padrões de impressão.

Sintaxe

SetPrint(cAlias, cNomeRel, cPerg, cDesc1, cDesc2, cDesc3, cDesc4, lDic, aOrdem, lComp, cClass)

Parâmetros

cAlias – Alias do Arquivo Principal (se existir)
cNomeRel – Nome padrão do relatório
cPerg – Nome do grupo de perguntas
cDesc1 ..cDesc4 – Descrição do Relatório
lDic – Habilita o Dicionário de Dados
.T. – Habilita (só utilizar em conjunto com a função ImpCadast)
.F. – Desabilita
aOrdem – Array contendo as ordens de indexação do arquivo principal.
lComp – Habilita a alteração da compressão do relatório
.T. – Habilita
.F. – Desabilita
cClass – Classificação do Relatório por Tamanho ("G","M" ou "P")
P – 80 colunas
M – 132 colunas
G – 220 colunas

Retorna

ExpC – Nome do arquivo com o relatório impresso em disco opcionalmente alterado pelo usuário

Comentários

Esta função possibilita a alteração de determinados padrões dos relatórios. Ela funciona em conjunto com a função SetDefault.

Exemplo

```
// Define Variáveis
cString:= "SB1"
NomeRel:= "MATR290"
cPerg := "MTR290"
titulo := "RELAÇÃO PARA ANÁLISE DOS ESTOQUES"
cDesc1 := "Este relatório demonstra a situação de cada item em "
cDesc2 := "relação ao seu saldo , seu empenho , suas entradas previstas"
cDesc3 := "e sua classe ABC."
aOrd := {" Por Codigo "," Por Tipo "}
Tamanho:= "G"
// Envia controle para a função SETPRINT
NomeRel := SetPrint( cString, NomeRel, cPerg, @titulo, cDesc1,;
cDesc2, cDesc3, .F., aOrd, .F., Tamanho )
If LastKey() = 27 .or. nLastKey = 27
RestScreen(3,0,24,79,cSavScr1)
Return
Endif
SetDefault(aReturn,cAlias)
If LastKey() = 27 .OR. nLastKey = 27
RestScreen(3,0,24,79,cSavScr1)
Return
Endif
```

SET FILTER

Esconde registros que nao atendam uma condição

Sintaxe

SET FILTER TO [<lcondição>]

Argumentos

<lcondição> , uma expressao lógica que define um conjunto específico de registros das área de trabalho corrente que sejam acess¡veis para processamento.

SET FILTER TO sem um argumento desativa a condição filtro.

Descrição

Quando uma condição FILTER está ativa, a área de trabalho corrente age como se contivesse somente os registros que atendem a condição especificada. Uma condição FILTER, uma das propriedades de uma área de trabalho. Uma vez ativada, a condição pode ser retornada na forma de uma cadeia de caracteres usando-se a função DBFILTER().

Exemplos

Este exemplo filtra somente aqueles registros onde a idade seja maior do que 50 no arquivo Employee.dbf:

```
USE Employee INDEX Name NEW
SET FILTER TO Age > 50
LIST Lastname, Firstname, Age, Phone
SET FILTER TO
```

SetRegua

Tipo: Impressão (DOS/ Windows)

Inicializa régua padrão em relatórios.

Sintaxe

SetRegua(nRegs)

Parâmetros

nRegs – Número de registros que serão processados.
Retorno
Nil

Exemplo

```
DbSelectArea("SA1")
SetRegua(LastRec())
While ( ! Eof() )
@ Li, 001 PSAY SA1->A1_NOME
DbSkip()
IncRegua()
End Do
```

Comentário
Ver também incRegra.

Somar

Tipo: Processamento

Faz o somatório de um arquivo, retornando o valor acumulado de um campo determinado.

Sintaxe

Somar(cAlias, cCond, cCampo)

Parâmetros

cAlias – Alias do arquivo
cCond – Condição para soma
cCampo – Campo a ser somado

Exemplo

Somar("SI1","I1_CLASSE='S'","I1_SALANT")
Caso o usuário não deseje definir nenhuma condição, a ExpC2 deve ser ".T.".

Anotações

Tabela

Tipo: Processamento

Devolve o conteúdo da tabela de acordo com a chave. Esta função é usada para a obtenção do conteúdo de uma determinada tabela, na chave especificada.

Retorna o conteúdo, possibilitando inclusive a exibição de um "help" caso a tabela não exista.

Sintaxe

Tabela(cTab,cChav,lPrint)

Parâmetros

cTab – Número da tabela a pesquisar (deve ser informado como caracter).
cChav – Chave a pesquisar na tabela informada.
lPrint – Indica se deve (.T.) ou não (.F.) exibir o help ou a chave NOTAB
se a tabela não existir.

Retorna

cRet – Conteúdo da tabela na chave especificada. Retorna nulo caso a tabela não exista ou a chave não seja encontrada.

Exemplo

```
// Exemplo de uso da funcao tabela:
// Suponha a existencia da tabela 99 (tabela de
// vendedor x Comissao):
// Chave Conteudo
// ————————————————————————————
// V0 10
// V1 2.20
// V3 5
// Pode-se fazer um gatilho que, quando da informacao do
// codigo do vendedor no cadastro, sugira o percentual
// da tabela acima, de acordo com as duas primeiras po-
// sicoes do codigo digitado:
//Gatilho-Dominio : A3_COD
// Cta. Dominio: A3_COMIS
// Regra : Val(Tabela("99",Left(M->A3_COD,2)))
```

TamSX3

Tipo: Processamento

Retorna o tamanho de um campo no SX3 (dicionário de dados).

Sintaxe

TamSx3(cCampo)

Parâmetros

cCampo – Nome do campo.

Retorna

aTam – Array com o tamanho e decimais do campo.

Exemplo

```
// Exemplo de uso da funcao TAMSX3

// Array auxiliar:

aCampos := { {"B1_COD" ,"C"},;
{"B1_DESC","C"},;
{"B1_QE" ,"N"},;
{"B1_PRV1","N"} }

// Cria arquivo de trabalho com o tamanho dos campos
```

```
// exatamente como na base de dados, evitando erros de ]

// "Data Width Error":

i := 0
aStru := {}
For i:=1 To Len(aCampos)
cCpo := aCampos[i,1]
cTp := aCampos[i,2]
aTam := TamSx3(cCpo)
aAdd(aStru,{cCpo,cTp,aTam[1],aTam[2])
Next i
cArq := CriaTrab(aStru,.T.)

// O programa continua...

Return
```

Texto

Tipo: Processamento

Não permite a digitação seguida de mais de um espaço em branco, em campo do tipo Caracter.

Sintaxe

Texto(ExpC)

Parâmetros

ExpC1 – Expressão a ser verificada

Exemplo

Texto()

@...TO

Tipo: Tela

Desenha um box 3d.

Sintaxe

@ nLInha1,nColuna1 TO nLinha2,nColuna2 <TITLE> cTítulo

Parâmetros

nLinha1 – Número da linha superior
nColuna1 – Número da coluna superior

nLinha2 – Número da linha inferior
nColuna2 – Número da coluna inferior
cTítulo – Titulo apresentado na linha superior (opcional)

Comentários

A cláusula TITLE é opcional. Se for omitida, o box não terá título.

Exemplo

```
@ 000, 000 TO 430, 500 DIALOG oDlg TITLE "Interpretador xBase for Windows"
@ 060, 005 TO 185, 245 TITLE 'Exemplos'
@ 070, 010 BUTTON "_Objetos B sicos" SIZE 70,20 ACTION Execute(BasicObj)
@ 070, 090 BUTTON "_Browses" SIZE 70,20 ACTION Execute(Browse)
@ 130, 170 BUTTON "Dlg c/Refresh " SIZE 70,20 ACTION Execute(DlgDinam)
@ 160, 090 BUTTON "SQL" SIZE 70,20 ACTION Execute(SqlDemo)
@ 192,218 BMPBUTTON TYPE 1 ACTION Close(oDlg)
ACTIVATE DIALOG oDlg CENTERED
```

TM

Tipo: Processamento

Devolve a picture de impressão de campos numéricos dependendo do espaço disponível.

Sintaxe

TM(nValor, nEdição, nDec)

Parâmetros

nValor – Valor a ser editado
nEdição – Espaço disponível para edição
nDec – Número de casas decimais

Retorna

ExpC1 – Picture ideal para edição do valor nValor.

Comentários

Esta rotina leva em consideração duas variáveis:

MV_MILHAR – Determina se deve haver separação de milhar;
MV_CENT – Número de casas decimais padrão da moeda corrente.

Para ajustar o valor passado (ExpN1) ao espaço disponível (ExpN2) o programa verifica se pode haver separação de milhar, neste caso, a rotina eliminará tantos pontos decimais quantos sejam necessários ao ajuste do tamanho. Caso não seja possível ajustar o valor ao espaço dado, será colocado na picture o caracter de estouro de campo «. O programa também ajusta um valor ao número de decimais (ExpN3), sempre imprimindo a quantidade de decimais passados no parâmetro.

Exemplo

```
Cabec(Título,Cabec1,Cabec2,NomeProg,Tamanho,nTipo)
Endif
li:=li+1
nSalAnt := nSaldoAtu-nCompras-nRecProd-nRecCons
@li,00 Say cTipAnt
@li,05 Say nSalAnt Picture TM(nSalAnt, 14)
@li,23 Say nCompras Picture TM(nCompras, 17, 3)
```

Se o conteúdo do campo nSalAnt for: 3.423.659.234,48 o valor será impresso como: 3423659.234,48
UPPER() Converte caracteres minúsculos para maiúsculos

Sintaxe

UPPER(<cString>) --> cStringMaiusc

Argumentos

<cString>, a cadeia de caracteres a ser convertida.

Descrição

UPPER() , uma função de tratamento de caracteres utilizada para converter todos os caracteres em um string para maiúsculos. Ela esta relacionada a LOWER(), que converte todos os caracteres em um string para minísculos. UPPER() est relacionada …s funções ISUPPER() e ISLOWER(), as quais determinam se um string começa com uma letra maiíscula ou miníscula.

UPPER() geralmente, utilizada para formatar cadeias de caracteres para fins de exibição. Ela pode, porém, ser usada para normalizar strings para fins de comparacões onde nao se diferencia maisculas de minsculas, ou para fins de INDEXação.

Exemplos

Os exemplos a seguir ilustram os efeitos de UPPER():

```
? UPPER("a string")              // Resulta: A STRING
? UPPER("123 char = <>")    // Resulta: 123 CHAR = <>
```

Vazio

Tipo: Processamento

Verifica se o campo está vazio.

Sintaxe

Vazio(cCampo)

Parâmetros

cCampo – Campo a verificar

Retorna

ExpL1 – Valor Lógico de Retorno (.T. ou .F.)

Comentários

Retorna .T. se ExpC1 estiver vazio.

Exemplo

@ 9,10 Get cCodigo Valid !Vazio(cCodigo)

X3Picture

Tipo: Processamento

Devolve a picture do campo de acordo com o dicionário de dados.

Sintaxe

X3Picture(cCampo)

Parâmetros

cCampo – Nome do campo a verificar a picture.

Retorna

ExpC1 – Picture do campo no dicionário de dados.

Comentários

Função geralmente usada para atribuir aos relatórios a efetiva picture de campos numéricos em relatórios.

Exemplo

```
cRel:=SetPrint(cAlias, cPrograma , , @cTitulo, cDesc1, cDesc2, cDesc3 ,.T., aOrd )
SetDefault(aReturn, cString)

While !EOF()

nLinha:=nLinha+1

@nLinha, 1 Say SB2->B2_QATU Picture X3Picture("B2_QATU")
```

XFilial

Tipo: Processamento

Retorna a filial utilizada por determinado arquivo. Esta função é utilizada para permitir que pesquisas e consultas em arquivos trabalhem somente com os dados da filial corrente, dependendo de se o arquivo está compartilhado ou não (definição que é feita por meio do ambiente Configurador).

É importante verificar que esta função não tem por objetivo retornar apenas a filial corrente, mas retorná-la caso o arquivo seja exclusivo. Se o arquivo estiver compartilhado, a função xFilial retornará dois espaços em branco.

Sintaxe

xFilial(cAlias)

Parâmetros

cAlias – Alias do arquivo desejado. Se não for especificado, o arquivo tratado será o da área corrente.

Retorna

cFilArq – Retorna a Filial para o arquivo desejado.

Exemplo

```
// Exemplo de uso da funcao xFilial:

// Supondo que a filial corrente seja a 01:

@ 10,10 Say xFilial("SB1")

// A linha acima ira imprimir "01" se o arquivo de

// produtos estiver exclusivo. Se estiver compartilhado

// imprimira " ".

// Usando em processamentos (Pesquisa e processa

// enquanto for a mesma filial):

dbSeek(xFilial()+mv_par01)

While !EOF() .And. xFilial() == SB1->B1_FILIAL

... Processamento
Enddo
Return
```

X3USO

Verifica se o campo está disponível para uso.

Sintaxe

X3Uso(cUsado, [nModulo])

Argumento	Obrigat.	Tipo	Descrição
cUsado	Sim	C	Conteúdo do campo X3_USADO a ser pesquisado
nModulo	Não	N	Numero do módulo, caso não seja informado será assumido como padrão o número do módulo corrente.

Descrição

Esta função retornará um valor lógico, se for uma campo usado verdadeiro (.T.), caso contrário falso (.F.).

Exemplo

```
User Function <nome-da-função>()
Local lUsado := .F.
DbSelectArea("SX3")
DbSetOrder(2)
DbSeek("A1_COD")
If X3Uso( SX3->X3_USADO )
lUsado := .T.
EndIf
Return lUsado
```

X3Picture

Retorna a mascara de um campo do dicionário de dados SX3.

Sintaxe

X3Picture(cCampo)

Argumento	**Obrigat.**	**Tipo**	**Descrição**
cCampo	Sim	C	Nome de um campo cadastrado no SX3

Exemplo

```
User Function <nome-da-função>( cCampo )

Local cPicture

cPicture := X3Picture( cCampo )

Return cPicture
```

X3Cbox()

Retorna o conteúdo de um campo tipo combo contido no dicionário de dados SX3

Sintaxe

X3CBox()
Descrição

Esta função retorna conteúdo do campo combo de acordo com o registro posicionado no SX3 e o idioma corrente.

Exemplo

```
User Function <nome-da-função>( )
Local cTitulo
Local cDescri
Local cCombo
dbSelectArea("SX3")
dbSetOrder(2)
If dbSeek( cCampo )
cTitulo := X3Titulo()
cDescri := X3Descri()
cCombo := X3Cbox()
EndIf
Return
```

Anotações

X3Descri()

Retorna o descrição de um campo contido no dicionário de dados SX3

Sintaxe

X3Titulo()

Descrição

Esta função retorna a descrição do campo de acordo com o registro posicionando no SX3 e o idioma corrente.

Exemplo

```
User Function <nome-da-função>( )
Local cTitulo
Local cDescri
Local cCombo
```

```
dbSelectArea("SX3")
dbSetOrder(2)
If dbSeek( cCampo )
cTitulo := X3Titulo()
cDescri := X3Descri()
cCombo := X3Cbox()
EndIf
Return
```

X3Titulo()

Retorna o título de um campo contido no dicionário de dados SX3

Sintaxe
X3Titulo()
Descrição
Esta função retorna o título do campo de acordo com o registro posicionado no SX3 e o idioma corrente.
Exemplo

```
User Function <nome-da-função>( )
Local cTitulo
dbSelectArea("SX3")
dbSetOrder(2)
If dbSeek( "A1_COD" )
cTitulo := X3Titulo()
EndIf
Return
```

Próximo Passo

Esperamos que você tenha conhecido e aprendido melhor sobre os principais recursos da Linguagem de Programação ADVPL.

Mantenha esta apostila como roteiro para seu trabalho diário.

Se tiver alguma sugestão para melhoria do nosso material, utilize o nosso e-mail:

microsigaeducacao@microsiga.com.br.

Teremos satisfação em recebê-la e analisaremos a viabilidade de aplicá-la ao nosso material.

Agora, o próximo passo é:

Aplicar o conteúdo apresentado à rotina de trabalho de sua empresa!

Número do Registro:
APLP10100807