

SUMÁRIO

INTERPRETADOR xBASE	5
Necessidades de Software	6
Necessidades de Hardware	6
PROGRAMAÇÃO	7
Estrutura Básica de Programas	7
Operadores	8
Funções e Procedures	8
Estruturas de Controle	9
Variáveis	9
Tipos de Dados	9
Macros	10
Arrays	10
Blocos de Código	10
Compilação pelo RDMAKE	11
Configuração no Menu	12
DIFERENÇAS ENTRE RDMAKE DOS E WINDOWS	13
Principais Diferenças	13
Um novo modo de programar a interface	14
Criando programas únicos	18
O que muda em relação a impressão	27
FUNÇÕES PARA O INTERPRETADOR xBASE	29
AbreExcl	30
Activate Dialog	31
Aleatorio	32
Avalimp	33
Aviso	35
AxCadastro	36
@ n1,n2 BmpButton	37
@... Bitmap... Size	38
@... To...Browse	39
@...Button	40
Cabec	41
CalcEst	43
CalcSaldo	44
Capital	45
CGC	46
@...CheckBox...Var	47
ChkFile	48
Close	50
CloseOpen	51
ClosesFile	52

@...ComboBox...Itens...Size	53
Comp3	54
Condicao	55
ConfirmSX8	56
Contar	57
ConvMoeda	58
Credito	59
CriaTrab	60
CriaVar	62
DataValida	63
Debito	64
DeComp3	65
@...To...Dialog.....	66
Digito11	67
DrawAdv3D	68
DrawAdvWindow	69
EANDigito	70
Entre	71
Estrut	72
Execute	74
ExistChav	75
ExistCpo	77
ExistIni	79
Extenso	80
FinNatOrc	81
FinNatPrv	82
FinNatRea	83
Formula	84
FuncaMoeda	86
@... GET	87
GetAdvFval	88
GetMV	90
GetSX8Num	91
GravaOrcado	93
Help	94
ImpCadast	95
IncRegua	96
IncProc	97
IndRegua	98
LetterOrNum	99
MarkBrowse	100
MBrowse	101
Media	102
MesExtenso	103
Modelo2	104
Modelo3	108
MontaF3	111
MovimCC	112
Movimento	113
MsGetVersion	114

MsgBox	115
@..To...MultiLine	116
NaoVazio	117
Negativo	118
Orcado	119
OrcadoCC	120
OpenFile	121
OurSpool	122
Pergunte	123
Periodo	124
Pertence	125
PesqPict	126
PesqPictQt	127
Posicione	128
Positivo	129
ProcRegua	130
ProxReg	131
@...Radio	133
RecLock	134
RecMoeda	136
RestArea	137
RetASC	138
RetIndex	139
RollBackSX8	140
RptStatus	141
Saldo	142
SaldoCC	143
SaldoCusto	144
SaldoSB2	145
SetDefault	146
SetDlg	148
SetPrint	149
SetRegua	151
SldBco	152
SldCliente	153
SldFornece	154
SldPagar	155
SldReceber	156
SomaContas	157
SomaMovim	158
Somar	159
SomaSaldo	160
SumMovimCC	161
Tabela	162
TamSX3	163
Texto	164
@ ...TO	165
TM	166
Variação	168
Vazio	169

X3Picture	170
XFilial	171
XMoeda	172
Funções para impressão de etiquetas	
padrão ZPL e Allegro	173
MSCBPRINTER	174
MSCBBEGIN	175
MSCBEND	176
MSCBSAY	177
MSCBSAYBAR	178
MSCBSAYMEMO	180
MSCBBOX	181
MSCBLineH	182
MSCBLineV	183
MSCBLOADGRF	184
MSCBGRAFIC	185
MSCBWRITE	186
Tipos de Fontes para Zebra:	187
Tipos de Fontes para Allegro:	188
RdMAKE EM AMBIENTE SQL	191
EXECUTANDO	193
Executando com Debug	193
Erros de Compilação	198
Erros e ações a serem tomadas	198
EXEMPLOS DE PROGRAMA	200
Referências e Abreviaturas	219



INTERPRETADOR xBASE

O interpretador de rotinas dinâmicas tem o objetivo de permitir ao usuário do SIGA Advanced construir seus próprios programas, agregá-los ao menu dos módulos e executá-los de uma forma transparente ao operador. De forma análoga, as rotinas escritas pelo usuário também podem fazer parte, por meio da função ExecBlock, do Arquivo de Fórmulas, das Planilhas, dos Gatilhos, das Validações e Inicializações no Dicionário de Dados, dos campos de Débito e Crédito, Histórico e Valor, do Arquivo de Lançamentos Padronizados e dos Pontos de Entrada, enfim, de todos os campos onde o SIGA aceita uma expressão que é interpretada em tempo de execução.

O usuário ainda tem a possibilidade de utilizar os recursos de programação adotados pelo SIGA Advanced através do uso de um grande número de suas FUNÇÕES. O objetivo principal deste manual é exatamente o de descrever e explicar o uso destas 180 Funções.

Um programa ou rotina para ser interpretada pelo SIGA Advanced necessita cumprir duas fases:

1. Ser compilado pelo RDMAKE, utilitário da MICROSIGA que gera os ponteiros de ligação de um programa .PRG, .PRW ou .PRX gerando o interpretável ._IX ou ._IW;
2. Ser configurado no menu do módulo com a # (cerquilha) precedendo o nome do programa, ou ser chamada via ExecBlock;

A fase de interpretação do programa pelo SIGA Advanced já é intrínseca a este e será acionada sempre que encontrar um # na primeira posição do nome do programa no menu.

Necessidades de Software

Para que o Programa seja compilado pelo RDMAKE é necessário que o ambiente que irá compilar o programa possua o ADVPL16 que encontra-se no CD da versão 4.06 do SIGA Advanced, já que o RDMAKE submete o programa (.PRG, .PRX ou .PRW) à apreciação do ADVPL16 antes de proceder a geração do interpretável.

Após compilado pelo RDMAKE, não é mais necessária a presença do ADVPL16 para a interpretação dentro do SIGA Advanced.

Necessidades de Hardware

O RDMAKE necessita de um mínimo de 450Kb de memória convencional para compilar um programa de 2048 linhas. Programas menores poderão ser compilados com uma quantidade de memória menor, até o limite do ADVPL16, aproximadamente 370Kb. O equipamento mínimo é um PC-XT de 640 Kb para a compilação. Quanto à interpretação pelo SIGA Advanced, ficam válidas as recomendações para este produto.

Estrutura Básica de Programas

Este manual de utilização do RDMAKE não tem a pretensão de ensinar código xBase, tão somente pretende mostrar as diferenças entre a programação XBase e o que é aceito pelo Interpretador de Programas do SIGA Advanced.

Por estar extremamente próximo do padrão de programação XBase, recomendamos que sejam consultadas literaturas próprias para o caso de aprendizagem da programação desta linguagem.

A estrutura de um programa escrito para ser interpretado pelo SIGA Advanced aderem às normas do padrão XBase, com as restrições que serão mostradas por tópico. O programa deve ser escrito em um editor de textos da preferência do usuário usando a seguinte estrutura básica:

- Corpo do Programa
- Chamadas de Funções Internas
- Chamadas de Funções Advanced
- Comentários
- Declaração de Funções no Programa

Operadores

São válidos todos os operadores do padrão XBase excetuando-se:

<i>Original</i>	<i>Substituir por</i>	
<code>++var</code>	- soma antes de executar	<code>var:=var+1</code> (antes do processo)
<code>--var</code>	- subtrai antes de executar	<code>var:=var-1</code> (antes do processo)
<code>var++</code>	- soma após executar	<code>var:=var+1</code>
<code>var--</code>	- subtrai após executar	<code>var:=var-1</code>
<code>var+=n</code>	- soma n a var	<code>var:=var+n</code>
<code>var-=n</code>	- subtrai n de var	<code>var:=var-n</code>
<code>var*=n</code>	- multiplica var por n	<code>var:=var*n</code>
<code>var/=n</code>	- divide var por n	<code>var:=var/n</code>
<code>var%=n</code>	- resto de var/n	<code>var:=var%n</code>
<code>var**=n</code>	- Exponencia var a n	<code>var:=var**n</code>

O operador de atribuição de igualdade deve obrigatoriamente ser `:=`

O operador de comparação deve obrigatoriamente ser `==` (duplo igual)

Funções e Procedures

As definições de funções para o Interpretador podem seguir as regras do padrão XBase, entretanto para o RDMAKE/Interpretador Procedures ou Funções são tão somente endereços de desvio do programa, razão que justifica as restrições a seguir:

- Não deve ser definido um nome de procedure ou function para a função principal. O interpretador xBase considera o ponto de entrada do programa na linha 1 do programa de usuário;
- Diretivas (`#include`, `#define`, etc...) não são aceitas no programa interpretável;
- Chamada de funções não permitem passagem de parâmetros. As variáveis de comunicação entre funções devem ser declaradas na função chamadora;
- Só podem ser definidas até 255 funções ou Procedures por programa;
- A declaração de variáveis `LOCAL` ou `PRIVATE` gera um erro no compilador;
- Chamadas recursivas a funções podem provocar erro de estouro de pilha ou ter um resultado inesperado, portanto são proibidas.

Estruturas de Controle

FOR...NEXT

DO...WHILE

EXIT e LOOP

IF..ELSEIF..ELSE...END

São estruturas aceitas sem quaisquer restrições pelo Interpretador.

Variáveis

Todas as variáveis do interpretador são PRIVATE e têm o tempo de vida válido por todo o programa, a partir de sua declaração.

Não são válidas declarações de variáveis com STATIC, LOCAL, PRIVATE e PUBLIC. Um erro será reportado pelo compilador nestes casos.

Tipos de Dados

Todos os tipos de dados do padrão XBase são aceitos no Interpretador, sem restrições.

São estes:

- Array
- Caractere
- Bloco de Código
- Numérico
- Data
- Lógico
- Memo
- NIL

Macros

São aceitas macros (&) dentro de programas interpretáveis, entretanto macros muito complexas ou inseridas no meio de uma string precisam ser revistas.

<code>b := &a</code>	- válida
<code>b:= var1&var2</code>	- precisa ser substituída por <code>b:= &("var1"+&var2)</code>
<code>&var:="Correta"</code>	- válida

Erros em macros só serão detectados em tempo de execução.

Arrays

O comportamento dos arrays no Interpretador e no padrão XBase são semelhantes, entretanto, na declaração destes arrays, o procedimento difere.

Declarações de arrays aceitas são:

```
aArray := { }  
aArray := ARRAY(5,2)
```

Declarações não aceitas:

```
DECLARE aArray[5]  
PRIVATE aArray[5][2]
```

Blocos de Código

Blocos de código serão avaliados perfeitamente pelo Interpretador.

Compilação pelo RDMAKE

Para que um programa possa ser interpretado pelo SIGA Advanced, este necessita ser compilado pelo programa RDMAKE que irá montá-lo em um formato que permite uma grande velocidade de interpretação e reestruturá-lo de modo que as interações e desvios de funções fiquem endereçadas dentro deste mesmo programa. O produto final da compilação será um arquivo com extensão `._IX` (DOS), `._IW` (WINDOWS) ou `._IX` (DOS e WINDOWS), estes arquivos não devem ser modificados pelo programador.

Os procedimentos de compilação seguem os seguintes passos:

- Editar o programa em um editor de textos de preferência do usuário;
- Submeter este programa à compilação do RDMAKE pela chamada

RDMAKE <nome do programa> -<diretiva>

O RDMAKE irá gerar:

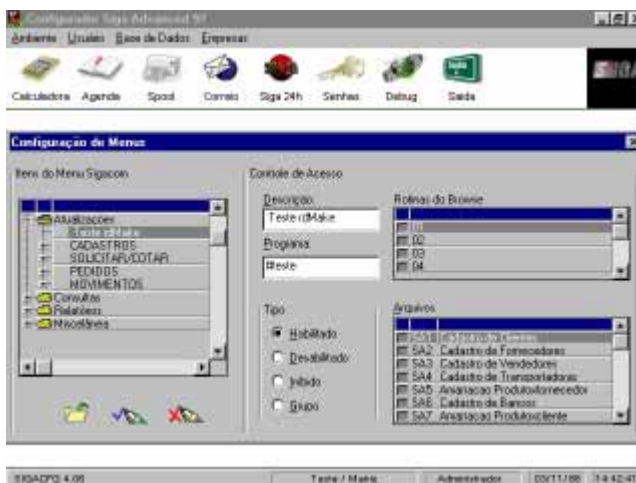
- sem diretiva: o arquivo interpretável `._IX` para versão DOS;
- com a diretiva `-W`: o arquivo interpretável `._IW` para versão Windows;
- com a diretiva `-X`: os arquivos interpretáveis `._IX` e `._IW` para ambas versões.

Os arquivos interpretáveis são gerados no diretório corrente. É indicado que o diretório corrente seja o mesmo do SIGA Advanced, já que o sistema chama o arquivo interpretável a partir deste diretório.

Configuração no Menu

O programa compilado deve ser incluído como uma opção no menu do módulo desejado. Isto é feito através da opção “Menu” do Módulo Configurador.

O SIGA Advanced saberá que o programa é interpretável através do caractere # colocado antes do nome do programa. Isto limita o nome de programa interpretável em sete posições.





DIFERENÇAS ENTRE RDMAKE DOS E WINDOWS

O SIGA Advanced é um sistema que possui versões para ambientes caracter, como o *DOS*, e ambientes gráficos, como o *Windows*. Muitos clientes ainda utilizam versões em ambiente *DOS*, e tem uma grande quantidade de específicos desenvolvidos apenas para esse ambiente. A conversão de todos estes programas de modo que fiquem compatíveis com as versões do *SIGA* em ambiente *Windows* não é realmente uma tarefa difícil. Porém, é uma ótima idéia gastar algum tempo para criar todos os programas de modo que trabalhem perfeitamente em qualquer um destes ambientes, economizando assim esforços em uma possível mudança de ambiente no futuro.

Veja a seguir algumas das principais dúvidas e dicas úteis para esta tarefa.

Principais Diferenças

O *Windows* é um ambiente gráfico e formado por objetos. Isto significa que todas as informações apresentadas na tela do microcomputador são apresentadas através de objetos, ou *componentes*. Um componente pode ser um botão, uma caixa de texto para a digitação de dados (o tão famoso *GET* que existia no *DOS*) e até mesmo uma janela (também chamada de *diálogo*). Diferentemente do ambiente *DOS*, onde pode-se exibir informações em qualquer lugar da tela e, em qualquer momento, no ambiente *Windows* todos os componentes devem pertencer a um componente denominado *Componente Pai*. Assim por exemplo, se você precisa criar uma janela para que o usuário digite uma certa informação, esta janela pertencerá à janela da aplicação, ou seja, à janela do *SIGA*. E da mesma forma, os botões, os *GETs*, e outros componentes pertencerão ao diálogo criado.

Desta forma, não existe a necessidade de “salvar” a área de tela sobre a qual a janela será exibida (e posteriormente restaurá-la), ou não se precisa indicar que a janela deverá ser exibida até que o usuário pressione a tecla *ESC*. A janela aparecerá no momento em que for ativada, e com ela todos os seus componentes. E somente será removida quando o usuário desejar fechá-la ou alguma instrução do código fizer o mesmo, levando consigo todos os componentes criados.

Percebe-se facilmente que a grande diferença entre o ambiente *DOS* e o ambiente *Windows* é a manutenção da interface com o usuário, sendo que o código de processamento é basicamente o mesmo.

Um novo modo de programar a interface

Apesar das diferenças basearem-se na interface, o novo paradigma de ambientes gráficos exerce uma mudança na velha forma de programar. Então, a lógica tradicional pode não ser mais a melhor opção na maioria dos programas, em relação a montagem da interface e no modo como o usuário irá acionar a execução dos processos. Por exemplo, imagine o seguinte código:

```
While .T.  
    // ... Monta a janela para a interface com o usuário  
  
    If LastKey() == 27  
        Exit  
    Endif  
EndDo  
  
// ... Executa o processamento com as informações recebidas...
```

Este código demonstra uma estrutura de fluxo em execução infinita, que somente deixará de executar quando o usuário pressionar a tecla *ESC*.

No ambiente *Windows*, os diálogos criados permanecem na tela até que o usuário deseje fechá-lo. Portanto, a mesma lógica seria algo como o demonstrado a seguir:

```
// ... Cria o diálogo, com todos os seus componentes, para interface com o usuário  
  
// ... Ativa o diálogo criado  
  
// ... Executa o processamento com as informações recebidas...
```

Neste caso, o processamento das informações somente seria executado após o usuário fechar o diálogo. Uma outra alternativa, que é a mais usada, podem ser a inclusão de dois botões, um indicando a confirmação e o outro indicando o cancelamento. Na verdade, o único botão que executa alguma coisa realmente é o botão de confirmação, pois o botão de cancelamento somente finaliza o diálogo e o processamento se encerra normalmente. Veja o exemplo:

```

//... Cria o diálogo com os componentes necessários

//... Dentre estes componentes existem dois botões:
//... Botão 1 -> A ação a ser executada ao pressionamento deste botão é, por exemplo, a
//... função XPTO
//... Botão 2 -> A ação a ser executada ao pressionamento deste botão é simplesmente o
//... fechamento do diálogo

//... Ativa o diálogo. O mesmo permanece na tela até o usuário fechar o diálogo, pressio-
//... nando o pequeno ícone do X no alto da janela ou pressionando o botão Cancelar, que
//... executa o código para o fechamento do diálogo.

//... Finaliza o programa. Caso o diálogo simplesmente seja fechado, o processamento
//... seguirá nesta linha, e o programa será finalizado sem que nada seja executado.

// ***** Função XPTO *****

//... Declaração da função XPTO. Esta função apenas será executada através do
//... pressionamento do botão Confirmar no diálogo do programa acima.

//... Executa o processamento com as informações recebidas.

//... Finaliza a função XPTO. Note que ao retornar da função, o diálogo permanece aberto
//... e, aos olhos do usuário, o processamento continua. Isso realmente acontece, a não ser
//... que nesta mesma função, antes de finalizar, seja executado o código indicando para
//... fechar o diálogo aberto no programa acima.

```

Para ilustrar tudo o que foi descrito, nas próximas páginas você verá exemplos de um programa que apresenta uma tela de interface com o usuário que indica a finalidade da rotina e permite-lhe cancelar ou confirmar o processamento. Estão demonstrados os códigos para as duas versões separadamente, *DOS* e *WINDOWS*, portanto, se você desejar testá-los crie os programas com a extensão *.PRG* para a versão *DOS* e *.PRW* para a versão *WINDOWS*. Mais à frente, demonstraremos como uni-los em um único arquivo com extensão *.PRX*, utilizando assim apenas um programa para as duas versões.

Versão DOS

```
//  
// Salva o conteúdo da tela atual na variável sTela.  
//  
//
```

sTela := SaveScreen(07, 06, 15, 71)

```
//  
// Desenha uma janela utilizando a função padrão do Microsiga.  
//  
//
```

DrawAdvWindow("Atenção!", 07, 08, 13, 71)

```
//  
// Apresenta as mensagens ao usuário.  
//  
//
```

```
sOldCor := SetColor("B/BG")  
@ 09, 10 Say "Este programa exige que os arquivos associados a ele estejam"  
@ 10, 10 Say "em modo exclusivo. Certifique-se de que nenhum outro usuário"  
@ 11, 10 Say "esteja usando o sistema neste momento."  
SetColor(sOldCor)
```

```
//  
// Esta é a parte do programa que permanece em execução até a confirmação ou o  
// cancelamento pelo usuário. Para isso se utiliza da função da Microsiga chamada MenuH.  
//
```

While .T.

```
NResp:=MenuH({"Ok", "Cancela"}, 13, 10, "b/w, w+/n, r/w", "OC", "", 1)  
If nResp == 1
```

```
//  
// Executa o processamento necessário, pois o usuário confirmou.  
//  
//
```

Endif

```
//  
// Indica a finalização do Loop  
// para finalizar o programa.  
//
```

Exit

EndDo

Return

Versão Windows

```
//  
//  
//  
//  
//  
//
```

Monta o diálogo para a interface com o usuário utilizando os comandos padronizados da Microsiga. Obs.: Todos os comandos aqui usados encontram-se demonstrados no programa chamado RDDEMO.PRW que acompanha todos os módulos do sistema.

```
@ 0,0 TO 135,360 DIALOG oDlg TITLE "Atenção!"
```

```
//  
//  
//  
//  
//
```

A partir de agora, todos os objetos indicados pertencerão ao diálogo acima. Note que você pode declarar todos os tipos de objetos, exceto um novo diálogo.

```
//  
//  
//  
//  
//
```

A linha abaixo define um *BOX*, ou seja, um quadro semelhante ao comando *@ TO* do *DOS*.

```
@ 01,02 TO 45,178
```

```
//  
//  
//  
//  
//
```

As linhas abaixo demonstram a exibição de mensagens na tela do usuário. O comando é exatamente o mesmo utilizado no ambiente *DOS*.

```
@ 10,10 Say "Este programa exige que os arquivos associados a ele estejam"  
@ 20,10 Say "em modo exclusivo. Certifique-se de que nenhum outro usuário"  
@ 30,10 Say "esteja usando o sistema neste momento." "
```

```
//  
//  
//  
//  
//  
//  
//  
//  
//  
//
```

A seguir, as linhas que demonstram o uso de botões. Note a cláusula *ACTION*, que indica que ação será executada ao pressionamento do botão. O botão do tipo 1 (que é o botão indicador de confirmação) executará a função *CONTINUA* e o botão do tipo 2 executará o encerramento do diálogo. Note que neste momento estamos apenas “definindo” os objetos que aparecerão no diálogo. Seu pressionamento e, conseqüentemente, a execução das ações somente serão possíveis no acionamento do diálogo.

```
@ 50,120 BMPBUTTON TYPE 1 ACTION Execute(Continua)  
@ 50,150 BMPBUTTON TYPE 2 ACTION Close(oDlg)
```

```
//  
//  
//  
//  
//  
//
```

Aciona o diálogo centralizado na tela. Repare que existe uma variável, chamada *oDlg*, que contém o objeto. Ela apareceu pela primeira vez na declaração do diálogo (mais acima) e é através dela que poderemos “manusear” o objeto (no caso o diálogo). Por exemplo, é necessária a variável para que a função *CLOSE* finalize o diálogo.

```
ACTIVATE DIALOG oDlg CENTERED
```

```
//  
//  
//  
//  
//
```

Aqui finaliza a execução do programa principal. O processamento somente chegará a este ponto quando o diálogo for fechado.

Return

Criando programas únicos

Do mesmo modo que os comandos mostrados para a criação de botões no exemplo para a versão *Windows*, a criação dos outros componentes é muito simples. Muitas pessoas têm dúvidas quanto ao acesso às informações destes componentes, mas isto também não é complicado.

O comando *GET* na versão *Windows* é idêntico ao utilizado na versão *DOS*, exceto pelo fato de que a cláusula *PICTURE* não pode ser abreviada e não existe a necessidade do comando *READ* para que as variáveis sejam editadas. Lembre-se: todos os componentes pertencem ao diálogo e somente serão exibidos, e editados, quando o diálogo for acionado.

Durante o processamento, que em nosso exemplo seria dentro da função *CONTINUA*, você pode acessar o conteúdo das variáveis editadas no diálogo, ou as variáveis referentes a qualquer outro componente criado pelos comandos padronizados do interpretador *RDMAKE* para *Windows*. Um exemplo completo destes comandos acompanha o sistema e chama-se *RDDEMO.PRW* (comentado no final do manual). É um programa que pode ser executado através do menu de qualquer módulo e utiliza-se de quase todos os comandos para o ambiente *Windows*.

Com tudo isto compreendido, ainda resta uma dúvida que atrapalha muitos programadores: como criar uma rotina única, que funcione para ambos os ambientes? O próximo exemplo demonstra o uso dos *GETs*, de alguns outros comandos do ambiente *Windows*, e como criar o programa de modo que ele possa ser executado em ambos os ambientes. Para isso, ele se utiliza de *diretivas do pré-compilador*.

Diretivas do pré-compilador são comandos que são executados exatamente antes da execução da própria compilação do programa. As diretivas que podem ser utilizadas no *RDMAKE* são: *DEFINE*, *IFDEF*, *IFNDEF* e *INCLUDE*. A diretiva *DEFINE* é utilizada para definir expressões constantes para o seu programa, a diretiva *INCLUDE* permite que você inclua um arquivo texto com um agrupamento de outras diretivas no seu programa, e as diretivas *IFDEF* e *IFNDEF* são diretivas condicionais.

As diretivas são sempre precedidas do caracter de código ASC 35 (#) e devem ser sempre escritas com os caracteres em caixa alta (com todas as letras maiúsculas). Por exemplo, imagine que você irá utilizar o valor de π (*PI*) para executar cálculos em seu programa. Ao invés de informar o valor 3,14 em todos os pontos onde ele será utilizado, você pode declarar um *DEFINE* e utilizar a expressão *PI* onde quiser que o cálculo seja executado. Algo como no exemplo:

```
#DEFINE      PI      3.14
nPerimetro := 2 * PI * nRaio
```

Definir expressões nestes casos é muito melhor do que utilizar variáveis. Porque antes da compilação, o pré-compilador analisa todo o programa e substitui a expressão *PI* pelo valor *3,14*. Somente então o programa é realmente compilado. Deste modo, não há consumo de recursos durante a execução do programa e a legibilidade do código-fonte é muito boa. E caso você deseje manter um arquivo texto com todas as suas definições e outras diretivas, você pode criá-lo com qualquer nome e inclui-lo no programa aonde as diretivas devem ser utilizadas:

```
// Conteúdo de DEFINIC.CH
#define      PI      3.14

// Conteúdo do Programa
#include "DEFINIC.CH"
nPerimetro := 2 * PI * nRaio
```

O padrão para estes arquivos de definições é a extensão CH, e o nome do arquivo deve estar sempre entre aspas duplas quando for incluído no seu programa.

Mas a grande vantagem destas diretivas não se encontra somente no fato de se poder criá-las. Você também pode verificar se elas já foram ou não definidas, através de diretivas condicionais.

Isto é muito útil quando você deseja, por exemplo, exibir uma mensagem na tela indicando um valor qualquer, pois você ainda está em fase de desenvolvimento do programa, mas quando for disponibilizá-lo a um usuário, você não desejará que o mesmo veja estas mensagens e não será necessário removê-las do código. Suponha que você crie a mensagem no seu código como o seguinte:

```
// código normal. . .

#ifdef LIGAAVISO
    Aviso("Debug","O valor da variável cVar é: "+cVar,{"Ok"})
#endif

// continuação do código. . .
```

Neste exemplo, você nota que não existe a definição da expressão *LIGAAVISO* e muito menos uma inclusão de um arquivo de definições. E como a diretiva condicional *IFDEF* verifica *se a expressão LIGAAVISO está definida*, a função *AVISO* não será executada. Na verdade, o que ocorre é o seguinte: Antes de compilar, o pré-compilador analisa o programa. Ele verifica que a expressão *LIGAAVISO* não está definida e, portanto, o código entre o *IFDEF* e o *ENDIF* não será compilado, ou seja, o código desta linha não entrará no arquivo interpretável (com a extensão *._IX* ou *._IW*) e, conseqüentemente, não será executado. Para fazer com que esta expressão esteja definida, não é necessário criá-la no código fonte ou em arquivos de inclusão. Basta defini-la no momento da compilação, utilizando a seguinte sintaxe na linha de comando do *RDMAKE*:



Se na linha de comando você utilizar: `RDMAKE NOMPROG -X /DLIGAAVISO`, a definição não funcionará. Porque `RDMAKE` executa um arquivo de lote (`RDMAKE.BAT`) que chama o verdadeiro compilador `RXMAKE.EXE` mas não passa o parâmetro adicional de definição para o mesmo.

Portanto, você pode habilitar ou desabilitar as mensagens do seu programa apenas mudando a maneira da compilação.

Estas explicações são necessárias para que você compreenda como o `RDMAKE` trata os programas que serão executados em ambas as versões do `SIGA`. Estes tipos de programas (que devem utilizar a extensão `PRX`, como já comentado) devem verificar a definição da expressão `WINDOWS`. Quando a mesma existir, você saberá que se trata de uma seção de programa que será executada na versão *Windows* do `SIGA`. Caso contrário, o programa será executado somente na versão *DOS* e não será necessário utilizar o parâmetro `/D` da linha de comando.

O que acontece é que quando o programa tem a extensão `.PRX` e você informa o parâmetro `-X` na linha de comando do `RDMAKE`, o código-fonte é compilado duas vezes. Uma para o ambiente *DOS* e outra para o ambiente *WINDOWS*. E na última compilação, que é a compilação para o ambiente *WINDOWS*, ele mesmo cria a definição da expressão `WINDOWS`. Veja o seguinte exemplo:

```
#IFDEF WINDOWS
    // Estas linhas só existirão na versão Windows
#else
    // Estas linhas só existirão na versão DOS
#endif
```

Então quando o compilador verifica o código-fonte pela primeira vez a expressão `WINDOWS` não existe, e qualquer código que esteja entre diretivas condicionais que checam a existência desta expressão simplesmente não serão compiladas. Na Segunda vez que o compilador verifica o código-fonte ele próprio definirá a expressão `WINDOWS`, e então o código que não será compilado será o código da cláusula `#ELSE`. Utilizando estes conceitos, os programas dos exemplos anteriores poderiam ser unificados da seguinte maneira:

```

//
// Declaração das variáveis utilizadas pelo programa.
//
//
lOk      := . T.
cDoc     := Space(6)
cSerie  := Space(3)
cMens   := Space(200)
cImpDup := "S"
nImpDup := 1
aImpDup := {"Sim", "Nao"}
aSeries := {}
aOp      := {"Confirma", "Redigita", "Abandona"}

//
// Monta um ARRAY com as séries disponíveis para o sistema.
//
//
dbSelectArea("SX5")
dbSeek(xFili al ("SX5")+"01")
While !EOF() . And. SX5->X5_TABELA == "01"
    aAdd(aSeries, Substr(AllTrim(SX5->X5_CHAVE), 1, 3))
    dbSkip()
EndDo

```

```

While . T.

    @12, 21 Say Space(29)                                Color "B/W"

    @07, 34 Get cDoc   Picture "@"
    @07, 49 Get cSerie Picture "@" When MontaF3("01") Valid
MontaF3()
    @08, 39 Get cMens  Picture "@!@S27"
    @10, 45 Get cImpDup Picture "!" Valid Pertence("SN")
    -
    //
    // Note o uso da função OKPROC caso o usuário confirme o processamento. O código da
    // confirmação poderia ser colocado diretamente dentro do CASE no lugar da chamada da função
    // OKPROC, mas como no Windows iremos utilizar um botão, devemos criar uma função para
    // executar este processamento.
    //
    nResp := MenuH(aOP, 12, 21, "b/w, w+/n, r/w", "CRA", "", 1)

Do Case
    Case nResp == 1 // Confirma
        OkProc()
        If !Ok
            Exit
        Else
            !Ok := . T.
            Loop
        Endif
    Case nResp == 2 // Redigita
        Loop
    Case nResp == 3 // Abandona
        Exit
EndCase

EndDo

//
// Aqui terminará o bloco do código que será considerado para o ambiente DOS. Como a seguir
// começará o bloco do ambiente WINDOWS (#ELSE), para a primeira compilação ele será
// desconsiderado até aonde encontra-se o #ENDIF. Portanto, para o ambiente DOS o programa
// termina aqui. O processamento principal somente será executado na confirmação pelo usuário (que
// acionará a função OKPROC).
//
//
// Início do bloco que será considerado na segunda etapa da compilação, a compilação para o
// ambiente WINDOWS. Os comentários anteriores são válidos. Portanto para a segunda etapa da
// compilação o programa começará aqui (exceto quanto a declaração das variáveis logo no início).
//
//
#ELSE

    @ 212, 148 To 418, 652 Dialog oDados Title "Dados da Nota Fiscal "
    @ 001, 001 To 100, 250

```

```
//  
//  
// Agora inicia-se a parte do programa que identifica as diferenças entre os ambientes. Lembre-se que  
// cada código somente será compilado em uma das etapas da compilação, dependendo da existência  
// ou não da definição WINDOWS.  
//
```

```
//  
// Na primeira etapa da compilação, que é o ambiente DOS, não estará definido a expressão  
// WINDOWS, então esta primeira parte do código será considerada. A parte do #ELSE, mais abaixo,  
// será descartada.  
//
```

#IFDEF WINDOWS

```
//  
//  
// Salva a área da tela aonde serão exibidos os dados e monta a tela no padrão DOS.  
//
```

```
_sTela := SaveScreen(05, 17, 14, 68)  
DrawAdvWindow("Dados da Nota Fiscal", 05, 19, 12, 68)  
@06, 20 TO 11, 67 Color "B/BG"  
@07, 21 Say "Nota Fiscal:" Color "B/BG"  
@07, 42 Say "Serie:" Color "B/BG"  
@08, 21 Say "Mensagem da nota:" Color "B/BG"  
@10, 21 Say "Imprimir as duplicatas?" Color "B/BG"
```

```
//  
//  
// Esta é a estrutura de fluxo infinita que geralmente se usa para permitir ao usuário redigitar, por  
// exemplo, no ambiente DOS. Assim os GET's serão executados até o usuário confirmar ou  
// abandonar o processamento.  
//
```

```
//
//
// O uso da função OEMTOANSI serve para compatibilizar os caracteres ANSI (padrão DOS) com o
// ambiente Windows, permitindo que alguns caracteres especiais sejam impressos (no caso do
// exemplo, o acento de “série”). O comando COMBOBOX monta uma lista Drop-Down com as
// séries obtidas no início do programa, e o comando RADIO disponibiliza as duas opções para escolha
// no padrão do ambiente Windows.
//
```

```
@ 009,007 Say OemToAnsi("Nota Fiscal:")
@ 009,149 Say OemToAnsi("Série")
@ 024,006 Say OemToAnsi("Mensagem da Nota:")
@ 080,008 Say OemToAnsi("Imprimir duplicatas?")

@ 009,038 Get cDoc Picture "@!" Size 100,100
@ 033,007 Get cMens Picture "@!" Size 230,040 MEMO

@ 009,168 ComboBox cSerie Items aSeries Size 50,50

@ 080,060 Radio aImpDup Var nImpDup

@ 083,172 BmpButton Type 1 Action Execute(OkProc)
@ 083,210 BmpButton Type 2 Action Close(oDados)
```

Activate Dialog oDados Centered

```
//
//
// Aqui termina o bloco considerado para o ambiente WINDOWS. O diálogo será mantido na tela até
// que o usuário pressione o botão do tipo 2, que fechará o mesmo, ou pressione o botão do tipo 1, que
// executará a mesma rotina da confirmação do ambiente DOS. Após o diálogo ser fechado, o
// processamento continuará após o #ENDIF e o programa será finalizado.
//
//
```

#ENDIF

```
//
//
// Aqui termina o programa principal. O processamento retornará para este ponto quando a função
// OKPROC terminar seu processamento ou quando o usuário cancelar. No caso do ambiente DOS,
// isto ocorrerá quando, por qualquer destes motivos, o loop infinito terminar, e no ambiente
// WINDOWS, ocorrerá quando o diálogo for fechado.
//
```

Return

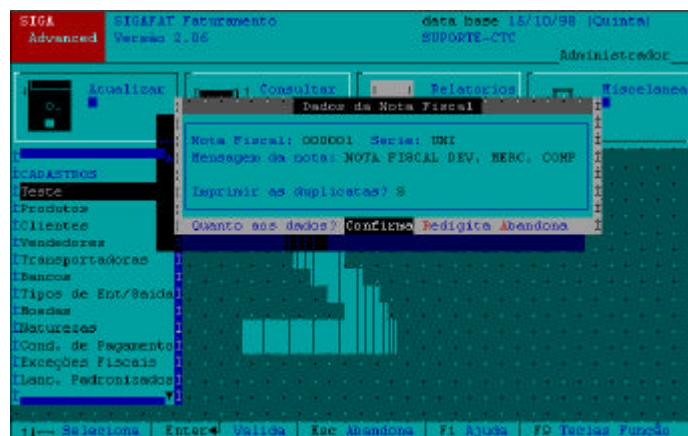


Figura 1 - Tela do programa de exemplo no ambiente DOS

```
//
//
// Esta é a função OKPROC. É a mesma que será executada em ambas as versões, portanto encontra-
// se fora dos blocos da checagem do ambiente (#IFDEF - #ELSE - #ENDIF), e estará em qualquer
// um dos arquivos interpretáveis gerados.
//
```

Function OkProc

```
//
//
// A primeira tarefa que a função executa é uma checagem dos dados digitados. Tanto no ambiente
// DOS como no ambiente WINDOWS, os valores estarão nas variáveis informadas nos GET's. O
// ambiente WINDOWS não tem o comando READ, mas as variáveis são disponibilizadas para
// edição a partir do momento em que o diálogo é ativado até o usuário pressionar o botão de
// confirmação. Note que até aqui, o diálogo da versão WINDOWS não foi fechado, de modo que
// quando a função detecta inconsistências e retorna, o processamento retornará para o diálogo aberto,
// e o usuário poderá reeditar as variáveis, cancelar ou confirmar novamente o processamento. No
// caso do ambiente DOS, foi criada a variável IOK que identificará, no programa principal, que o
// loop infinito deverá continuar, pois o processamento não foi concluído.
//
//
//
```

```
    Aviso("Erro!", "Voce deve informar o numero de um documento de nota fiscal e
serie existente.", {"0k"})
```

```
    Iok := .F.
```

```
    Return
```

```
Endif
```

```
dbSelectArea("SF2")
```

```
dbSetOrder(1)
```

```
If !dbSeek(xFili al ("SF2")+cDoc+cSerie)
```

```
    Aviso("Erro!", "Número do documento/Série não encontrado.", {"0k"})
```

```
    Iok := .F.
```

```
    Return
```

```
Endif
```

```
//
//
// Aqui seria executado o processamento com a nota fiscal informada. Para verificar a escolha do
// usuário, por imprimir ou não as duplicatas, deve-se tratar os dois ambientes diferentemente. Porque
// no caso do ambiente DOS utilizou-se uma variável em que o usuário digitava "S"im ou "N"ão e no
// ambiente WINDOWS, utilizou-se o comando RADIO que altera uma variável numérica, contendo
// o número da opção escolhida.
//
```

```
#IFDEF WINDOWS
```

```
    If cImpDup == "S"
```

```
        // Executa o processamento para impressao das duplicatas. . .
```

```
    Endif
```

```
#ELSE
```

```
    If nImpDup == 1
```

```
        // Executa o processamento para impressao das duplicatas. . .
```

```
    Endif
```

```
#ENDIF
```

```
//
//
// Aqui encontra-se o final da função OKPROC, aonde o processamento já foi concluído. A variável
// IOK, utilizada como Flag pelo código para o ambiente DOS já continha desde o início o valor
// verdadeiro (.T.), portanto no código deste ambiente o processamento será finalizado (vide o código
// do início do programa, na linha após a chamada desta função). Para o código do ambiente
// WINDOWS, so nos resta fechar o diálogo. A condição a seguir faz esta verificação e o
// processamento retorna para o programa principal.
//
//
```

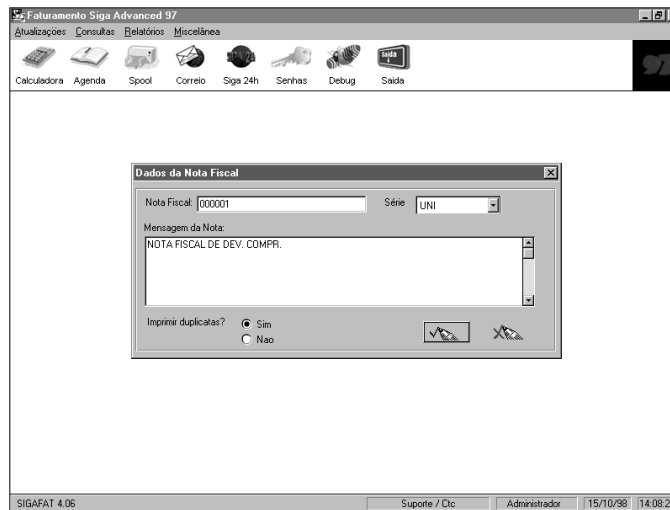


Figura 2 - Tela do programa de exemplo no ambiente WINDOWS

O que muda em relação a impressão

Quanto a impressão, as mudanças também são pequenas mas necessárias. Apesar de no ambiente Windows existirem *Drivers* para as impressoras instaladas, o SIGA não se utiliza deles na maioria das vezes. Quando o usuário escolhe uma impressora qualquer da lista exibida na interface padrão de relatórios (criada pela função *SETPRINT*), o sistema imprime diretamente para a porta definida, mas se o usuário escolher nesta lista “Default do Windows”, o sistema então imprimir através do driver do Windows. É por esse motivo que ao imprimir em qualquer impressora que não seja o “Default do Windows”, o sistema requer que a impressora esteja local ou que uma porta esteja capturada para a impressora da rede.

Tudo isto influi na inicialização da impressora. No ambiente DOS, bastava utilizar os comandos *SET PRINTER TO* e *SET DEVICE TO* para preparar a impressão, mas no ambiente *WINDOWS* deve-se ainda preparar a porta para a impressão, que será desviada da impressora padrão do sistema operacional. Como na maioria dos relatórios, utilizamos as funções padronizadas da MICROSIGA, como a *SETPRINT* e a *SETDEFAULT*; isto é transparente para o programador. O sistema se encarrega dessas preparações. Mas caso você não se utilize destas funções, deverá inicializar a porta desejada ou, então, a impressão não será possível. Neste caso, utilize a função *INITPRINT*, como no exemplo:

```

SET PRINTER TO LPT1
SET DEVICE TO SCREEN

#ifdef WINDOWS
    INITPRINT( )
#endif

```

Note que a função *INITPRINT* somente é necessária para o código do ambiente *WINDOWS*.

Para finalizar o acesso à porta, deve-se utilizar a função *MS_FLUSH*. Esta função realiza uma série de tarefas e é absolutamente necessária ao final do programa, pois caso não seja utilizada, ocorrerão erros na próxima tentativa de impressão. Também é utilizada no código para ambiente DOS, portanto não deve estar entre diretivas condicionais.

O que resta então é a forma como a impressão é feita. Programadores tradicionais do padrão xBASE estão acostumados a utilizar o comando *SAY* para a impressão. Porém, no ambiente *WINDOWS* existe um novo comando: *PSAY*.

No ambiente *WINDOWS*, o comando *SAY* sempre direciona a saída para a tela, independente da ativação do comando *SET DEVICE TO*. Utiliza-se o comando *PSAY* para desviar a saída para a impressora. Isto deve-se ao fato de que neste ambiente, os dois dispositivos, tela e impressora, são tratados separadamente. Assim, você não precisa retornar a saída para a tela, usando *SET DEVICE TO SCREEN*, durante uma impressão se desejar exibir informações ao usuário.

É muito comum em programas únicos desenvolvidos para os dois ambientes que se crie todo o código utilizando *PSAY*, porém é incluída a diretiva como no exemplo a seguir, que irá substituir este comando pelo padrão do ambiente *DOS* (*SAY*) durante a pré-compilação para este ambiente.

Tendo estes conceitos em mente, ficará bem mais fácil para o programador desenvolver programas *RdMAKE* para o ambiente *WINDOWS* ou compatibilizar os programas existentes. Na grande maioria das vezes, o trabalho envolvido na compatibilização de programas existentes para o ambiente *DOS* deve-se à lógica empregada ou a complexidade das telas criadas. E deve-se lembrar que alguns conceitos são extremamente diferentes entre os dois ambientes. No *DOS*, as coordenadas estão em caracteres, nas linhas por colunas, e no *WINDOWS*, as mesmas coordenadas se encontram em *PIXELS* (pontos gráficos da tela do microcomputador). Isto dificultará o posicionamento dos diálogos e componentes, pois as alterações podem influir pouco ou muito na aparência final e portanto o programador deve executar um grande número de alterações e testes no código-fonte até que o resultado final seja o esperado.

E o mais importante é que devemos sempre procurar desenvolver aplicações que já estejam preparadas para o ambiente *WINDOWS* (caso usemos ainda o ambiente *DOS*) ou desenvolvê-las de modo que uma customização futura seja facilitada.



FUNÇÕES PARA O INTERPRETADOR xBASE

A seguir são apresentadas as funções SIGA Advanced 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 o reabre 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 obrigatoriamente sua estrutura definida no SX3.

Exemplo

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

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

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
```


Avalimp

Tipo: Relatórios

Configura a impressora através 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 (respectivamente 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:
#ifdef 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

```

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
```

AxCadastro

Tipo: Processamento

Geração de modelo 1. Rotina para criação e manutenção de cadastros no padrão do SIGA Advanced, contendo as opções padronizadas: PESQUISA, INCLUSÃO, ALTERAÇÃO, VISUALIZAÇÃO e EXCLUSÃO.

Disponibiliza o Browse e todas as funções de cadastramento padrão.

Sintaxe

AxCadastro(cAlias,cTitulo,cDel,cOk)

Parâmetros

- cAlias – Alias do arquivo. Deve obrigatoriamente ter sua estrutura definida no SX3.
- cTitulo – Título da Janela.
- cDel – Função para validar a exclusão.
- cOk – Função para validar a Inclusão/Alteração.

Comentários

Deve ser utilizada para editar arquivos específicos (Família SZ?), sendo semelhante aos cadastros de Clientes, Fornecedores e etc...

Exemplo

```
// Exemplo de uso de cadastro de arquivo específico:  
AxCadastro("SZ1","Cadastro de Descontos",".T.",".T.")  
Return
```

@ 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.

@... Bitmap... Size

Tipo: Tela Windows

Define a área em que será mostrado um BITMAP na janela.

Sintaxe

@ nLinha,nColuna BITMAP SIZE nAltura,nLargura FILE cArq

Parâmetros

- nLinha – Número da Linha superior
- nColuna – Número da Coluna superior
- nAltura – Altura de apresentação do BITMAP
- nLargura – Largura de apresentação do BITMAP
- cArq – Nome do arquivo BITMAP

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

Parâmetros

- 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).

@...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.

Cabec

Tipo: Impressão

Esta função imprime, na impressora selecionada, o cabeçalho padrão dos relatórios do SIGA Advanced. 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

```
cTítulo := "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

```

CalcEst

Tipo: Processamento e Planilha

Devolve a quantidade e saldos do estoque de um produto/almojarifado em uma determinada data. Esta função é utilizada para a obtenção dos saldos iniciais em estoque na data em referência.

Sintaxe

CalcEst(cProduto, cAlmox, dData)

Parâmetros

- cProduto – Produto a ser pesquisado
- cAlmox – Almojarifado a pesquisar
- dData – Data desejada

Retorna

- aArray – Array contendo:
 - Elemento 1 - Quantidade inicial em estoque na data
 - Elemento 2 - Custo inicial na data na moeda 1
 - Elemento 3 - Custo inicial na data na moeda 2
 - Elemento 4 - Custo inicial na data na moeda 3
 - Elemento 5 - Custo inicial na data na moeda 4
 - Elemento 6 - Custo inicial na data na moeda 5
 - Elemento 7 - Quantidade inicial na segunda unidade de medida

Exemplos

```
aSaldos:=CalcEst(SB1->B1_COD,SB1->B1_LOCPAD, dDataBase)  
nQuant:=aSaldos[1]
```

CalcSaldo

Tipo: Processamento e Planilha

Calcula o saldo atual de uma determinada conta contábil até um determinado período. A conta deve estar posicionada no arquivo “SI1” ou “SI7”, de acordo com a moeda, antes da chamada desta função.

Sintaxe

CalcSaldo(nPer,nMoeda,lSalAnt)

Parâmetros

- nPer – Período (1 a 17) que será usado como limite para o cálculo de saldo.
- nMoeda – Moeda para o cálculo. Se não informada, é assumida a moeda 1.
- lSalAnt – Indica se deve (.T.) ou não (.F.) considerar o saldo inicial da conta. Se não informado, é assumido verdadeiro (.T.).

Retorna

- nSld – Retorna o Saldo atual da conta na moeda desejada.

Exemplos

```
// Exemplo de uso da função CALCSALDO:  
cConta := SA1->A1_CONTA  
dbSelectArea("SI1")  
dbSeek(xFilial("SI1")+cConta)  
? "Saldo atual da conta "+cConta+": "  
? CalcSaldo(Periodo())  
Return
```

Capital

Tipo: Processamento

Transforma as letras iniciais em Maiúsculas e as demais em Minúsculas.

Sintaxe

Capital(cTexto)

Parâmetros

cTexto – Texto a ser convertido

Exemplo

```
cCapital:=Capital("TEXT0 MAIUSCULO")  
// 0 retono será "Texto Maiúsculo"
```

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 sequência 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. Ex.:

“Atualiza Estoque” - .T./.F. = Marcada/Desmarcada

“Gera Duplicata” - .T./.F. = Marcada/Desmarcada

“Calcula IPI” - .T./.F. = Marcada/Desmarcada

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 compartilhado. 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 !ok
        // 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)
```

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
```

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 selecionada.

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.

Comp3

Tipo: Processamento

Compacta um valor numérico em uma string binária COMP-3. Esta função é utilizada para compactar a movimentação de Clientes e Fornecedores.

A descompactação é feita pela função complementar **DeComp3**.

Sintaxe

Comp3(aArray)

Parâmetros

aArray – Array contendo os 25 campos compactados (o Saldo Inicial, os 12 Débitos e 12 Créditos)

Retorna

ExpC1 – Nome do Campo aglutinador da compactação.

Exemplos

```
aSaldos:=DeComp3(A1_MOVIMEN)
nSaldo:=aSaldos[1]
For j := 2 To 24 STEP 2
    nSaldo := nSaldo - aSaldos [ j ] + aSaldos [ j + 1 ]
Next j
aSaldos[1]:=nSaldo
AFILL(aSaldos,0,2,24)
RecLock("SA1")
REPLACE A1_MOVIMEN With Comp3(aSaldos)
```

Ver também

Função DeComp3

Condicao

Tipo: Processamento

Esta função permite avaliar uma condição de pagamento, retornando um array multidimensional com informações referentes ao valor e vencimento de cada parcela, de acordo com a condição de pagamento.

Sintaxe

Condicao(nValTot,cCond,nVIPI,dData,nVSol)

Parametros

- nValTot – Valor total a ser parcelado
- cCond – Código da condição de pagamento
- nVIPI – Valor do IPI, destacado para condição que obrigue o pagamento do IPI na 1ª parcela
- dData – Data inicial para considerar

Retorna

- aRet – Array de retorno ({ { VALOR, VENCTO } , ... })

Exemplo

```
// Exemplo de uso da funcao Condicao:
nValTot := 2500
cCond := "002" // Tipo 1, Duas vezes
aParc := Condicao(nValTot,cCond,,dDataBase)
? "1| Parcela: "+Transform(aParc[1,1],"@E 9,999,999.99")
? " VencTo: "+DTOC(aParc[1,2])
? ""
? "2| Parcela: "+Transform(aParc[2,1],"@E 9,999,999.99")
? " VencTo: "+DTOC(aParc[2,2])
inkey(0)
Return
```

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.

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")

ConvMoeda

Tipo: Processamento

Converte o valor informado para a moeda selecionada.

Sintaxe

ConvMoeda(dData1,dData2,nValor,cMoeda)

Parâmetros

- dData1 – Data de emissão
- dData2 – Data de vencimento
- nValor – Valor a ser convertido
- cMoeda – Para qual moeda deverá converter

Retorna

- ExpN1 – Valor convertido (devolvido pela função)

Comentários

Esta função converte o valor definido por nValor para a moeda especificada em cMoeda na data dData.

A data dData2 não é utilizada.

O valor nValor é considerado em moeda 1.

Exemplos

```
nValor2 := ConvMoeda( D2_EMISSAO, , D2_TOTAL, cMoeda )  
nValor1 := ConvMoeda( D1_DTDIGIT, , D1_TOTAL, cMoeda )
```

Credito

Tipo: Processamento

Devolve o valor a crédito de uma determinada conta.

Sintaxe

Credito(cConta, nMês, nMoeda)

Parâmetros

- cConta – Código da Conta
- nMês – Mês do movimento desejado
- nMoeda – Moeda desejada para obtenção do valor a crédito

Exemplo

Credito("11103",03,1)

Ver também

Funções Debito, Saldo e Movimento

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
```

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
```

Debito

Tipo: Processamento

Devolve o valor a débito de uma determinada conta.

Sintaxe

Debito(cConta, nMês, nMoeda)

Parâmetros

- cConta – Código da Conta
- nMês – Mês do movimento desejado
- nMoeda – Moeda desejada para obtenção do valor a débito

Exemplo

Debito("11103",03,1)

Ver também

Funções Credito, Saldo e Movimento

DeComp3

Tipo: Processamento

Descompacta uma string binária COMP-3 em um array. Esta função é utilizada para descompactar a movimentação de Clientes e Fornecedores.

A compactação é feita pela função complementar **Comp3**.

Sintaxe

DeComp3(cCampo)

Parâmetros

cCampo – Nome do Campo aglutinador da compactação.

Retorna

aArray – Array contendo os 25 campos compactados.

Exemplo

```
aSaldos:= DeComp3(A1_MOVIMEN)
nSaldo := aSaldos [ 1 ]
For j:= 2 To 24 STEP 2
    nSaldo:= nSaldo - aSaldos [ j ] + aSaldos [ j + 1 ]
Next j
aSaldos[1]:=nSaldo
AFILL(aSaldos,0,2,24)
Reclock("SA1")
REPLACE A1_MOVIMEN With Comp3(aSaldos)
```

Ver também

Função Comp3

@...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 – Título da Janela

Comentários

Deve ser utilizada sem conjunto com o comando `ACTIVATE DIALOG`.

Exemplo

Ver exemplo no programa `RDDEMO` apresentado no final deste Manual.

Digito11

Tipo: Processamento

Cálculo de dígito verificador em módulo 11.

Sintaxe

Digito11(cCalc,cDigito)

Parâmetros

- cCalc – String para calcular o dígito
- cDigito – Dígito de verificação

Retorna

- ExpL1 – Valor Lógico de Retorno (.T. ou .F.)

Comentários

Esta função calcula o dígito de verificação de cCalc e o compara com cDigito, verificando a consistência.

Exemplos

```
@ 9, 10 Get cCodigo;  
Valid Digito11( SubStr(cCodigo, 1, 5), Substr(cCodigo, 6, 1))
```

DrawAdv3D

Tipo: Tela DOS

Desenha uma janela DOS em 3 dimensões.

Sintaxe

DrawAdv3D(cTitle, nLinha1, nColuna1, nLinha2, nColuna2, cCorFrente, cCorFundo, cStyle)

Parâmetros

- cTitle – Título da Janela
- nLinha1 – Número da linha superior
- nColuna1 – Número da coluna superior
- nLinha2 – Número da linha inferior
- nColuna2 – Número da coluna inferior
- cCorFrente – Cor da letra
- cCorFundo – Cor do fundo
- cStyle – R (onde R = Raised - define a impressão de baixo relevo. O padrão é alto relevo)

Exemplo

```
//  
DrawAdv3D("Janela 3D",01,24,20,24,B+,N,R)
```

DrawAdvWindow

Tipo: Tela DOS

Desenha uma janela padrão de acordo com a posição relativa passada como parâmetro. Para versão DOS.

Sintaxe

DrawAdvWindow(cTitulo,nLinha1,nColuna1,nLinha2, nColuna2)

Parâmetros

- cTitulo – Título da janela
- nLinha1 – Linha superior da janela
- nColuna1 – Coluna esquerda da janela
- nLinha2 – Linha inferior da janela
- nColuna2 – Coluna direita da janela

Comentários

Esta função desenha uma janela com o título cTitulo, com o canto superior esquerdo na posição nLinha1, nColuna1 e canto inferior direito na posição nLinha2, nColuna2.

Exemplos

DrawAdvWindow("Títulos em Aberto",3,4,20,76)

EANDigito

Tipo: Processamento

Calcula o dígito de controle para o código EAN usado em códigos de barras.

Sintaxe

EanDigito(cCod)

Parâmetros

cCod – Código de barras para o cálculo do dígito. É obrigatório o tamanho de 12 dígitos.

Retorna

cEan – Retorna o código EAN de 13 dígitos, sendo que a última posição é o dígito calculado.

Exemplo

```
// Exemplo de uso da funcao EANDIGITO:
// Gatilho
// Dom.: B1_CODBAR
// CtaDom.: B1_CODBAR
// Regra: Eandigito(PADL(AllTrim(M->B1_CODBAR),12,"0"))
// Ou usado em um programa rdmake:
cCod := EanDigito(PADL(AllTrim(M->B1_CODBAR),12,"0"))
cDig := Substr(cCod,12,1)
Alert("O digito calculado e': "+cDig)
Return
```

Entre

Tipo: Processamento

Verifica se o conteúdo do campo está entre o conteúdo de duas expressões (ExpX1 <= cCampo <= ExpX2). Se verdadeiro, retorna .T..

Usado normalmente em validações de campos digitados.

Sintaxe

Entre(ExpX1,ExpX2,cCampo)

Parâmetros

- ExpX1 – Expressão a comparar >=
- ExpX2 – Expressão a comparar <=
- cCampo – Nome do Campo

Retorna

- ExpL1 – Valor Lógico de Retorno (.T. ou .F.)

Exemplos

```
:  
If Entre("A1","A9",cSerie)  
    @ 5,10 Say "Serie Ok"  
Else  
    @ 5,10 Say "Serie Invalida"  
    Loop  
EndIf  
:
```

Estrut

Tipo: Processamento

Função para obtenção da estrutura de um produto previamente cadastrada no SG1 através dos Módulos “SIGAEST” ou “SIGAPCP”.

Sintaxe

Estrut(cProduto)

Parâmetros

cProduto – Código do produto PAI da estrutura.

Retorna

aStru – Retorna o array com a estrutura do produto na seguinte sintaxe:
{ { N° , Código , Comp. , Qtd. , TRT }, ... , ... }



Esta função requer que seja declarada a variável chamada “nEstru” contendo 0, para o seu correto funcionamento.

Exemplo

```
// Exemplo de uso da funcao Estrut:
cPrd      := Space(15)
aStru     := {}
nEstru    := 0
While .T.
    @ 10,10 Say "Codigo do Produto: "
    @ 10,30 Get cPrd Picture "@" Valid(!Empty(cPrd))
    Read
    If LastKey() == 27
```



```

        Exit
    Endif
    If !ExistCpo("SB1",cPrd)
        Loop
    Endif
    i:= 0
    aStru := Estrut(cPrd) // Obtem a estrutura
    nLin := 5
    For i := 1 To Len(aStru)
        @nLin,00 Say "Comp.: "+aStru[i,3]
        @nLin,20 Say "Quant.: "+Str(aStru[i,4],15)
        nLin := nLin + 1
        If nLin > 23
            @24,00 Say "Tecle <ENTER>..."
            Inkey(0)
            nLin := 5
        Endif
    Next i
EndDo

```

Execute

Tipo: Processamento

Executa uma Função definida em um mesmo .Pr? nas cláusulas <ACTION> ou <VALID>.

Sintaxe

Execute (cFunção)

Parâmetro

cFunção – Função a ser executada.

Exemplo

```
@ 75,098 BmpButton type 01 action Execute(S500IMP)
```

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,
// definida atraves do Configurador:
// Campo -> B1_COD
// Validacao do Usuario -> ExistChav("SB1")
// Ou em um Rdmake:
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
```

ExistCpo

Tipo: Processamento

Verifica se determinada chave existe no Alias especificado. Função utilizada em processamentos onde o código informado deve existir em determinado cadastro, na 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 alias.

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
```

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 funcao ExistIni:  
// Se existir inicializador no campo B1_COD:  
If ExistIni("B1_COD")  
    // Chama o inicializador:  
    cCod := CriaVar("B1_COD")  
Endif  
Return
```

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,IQtd,nMoeda)

Parametros

- nValor – Valor a ser gerado o extenso.
- IQtd – 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(Extensio(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
```


FinNatOrc

Tipo: Processamento e Planilha

Retorna o valor orçado da natureza.

Sintaxe

FinNatOrc(cNatureza,nMês,nMoeda)

Parâmetros

cNatureza – Natureza a ser pesquisada
nMês – Mês para cálculo
nMoeda – Moeda de saída

Exemplo

FinNatOrc("R001",10,1)

FinNatPrv

Tipo: Processamento e Planilha

Retorna o valor previsto de cada natureza.

Sintaxe

FinatPrv(cNatureza,dData1,dData2,nMoeda)

Parâmetros

cNatureza – Natureza a ser pesquisada
dData1 – Data Inicial para cálculo
dData2 – Data Final de cálculo
nMoeda – Moeda de saída

Exemplo

FinNatPrv("R001",CtoD("01/01/98"),dDataBase,1)

FinNatRea

Tipo: Processamento e Planilha

Retorna o valor realizado da Natureza.

Sintaxe

FinNatRea(cNatureza,dData1,dData2,nMoeda)

Parâmetros

cNatureza – Natureza a ser pesquisada

dData1 – Data Inicial para cálculo

dData2 – Data Final de cálculo

nMoeda – Moeda de saída

Exemplo

```
FinNatRea("R001",CtoD("01/01/98"),dDataBase,1)
aAdd(aL,"+-----+")
aAdd(aL,"|XXXXXXXX Relatorio de Teste   Pagina: XXXXX|")
aAdd(aL,"+---+-----+---+")
aAdd(aL,"| CODIGO   | DESCRICAO                |   PRECO |")
aAdd(aL,"+---+-----+---+")
aAdd(aL,"| XXXXXXX | XXXXXXXXXXXXXXXXXXXXXXXX | XXXXXX |")
aAdd(aL,"+---+-----+---+")
nLim:= 80 // Relatorio de 80 colunas
nLi:= 60
nPg:= 1
dbSelectArea("SB1")
dbGoTop()
While !EOF()
    If nLi > 55
        nLi := 0
        FmtLin({},aL[1],,,@nLi,.T.,nLim)
        FmtLin({dDataBase,nPg},aL[2],,,@nLi,.T.,nLim)
        FmtLin({},aL[3],,,@nLi,.T.,nLim)
        FmtLin({},aL[4],,,@nLi,.T.,nLim)
        FmtLin({},aL[5],,,@nLi,.T.,nLim)
        nPg := nPg + 1
    Endif
    aDados := {}
    aAdd(aDados,Subs(B1_COD,1,7))
    aAdd(aDados,Subs(B1_DESC,1,22))
```

Formula

Tipo: Processamento

Interpreta uma fórmula cadastrada. Esta função interpreta uma fórmula, previamente cadastrada no Arquivo SM4 através do Módulo Configurador, e retorna o resultado com tipo de dado de acordo com a própria fórmula.

Sintaxe

Formula(cFormula)

Parâmetros

cFormula – Código da fórmula 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.
```

```
// Fórmula cadastrada no SM4:  
// Código: F02  
// Regra : (GETMV("MV_JUROS")/100)+1  
// Ao usar esta fórmula, pode-se dar um acréscimo em um  
// valor de acordo com a taxa de juros cadastrada no parâmetro MV_JUROS:  
nValor := nValor * Formula("F02")
```

FuncaMoeda

Tipo: Processamento

Retorna um array contendo o valor do titulo em até cinco (5) moedas.

Sintaxe

FuncaMoeda(dData,nValor,nMoeda)

Parâmetros

- dData – Data utilizada como referência
- nValor – Valor utilizado como referência
- nMoeda – Moeda em que o valor se encontra

Retorna

- aRet – Array contendo o valor informado nas cinco moedas
({ nVal1,nVal2,nVal3,nVal4,nVal5 })

Exemplo

```
// Exemplo do uso da funcao FuncAMoeda:
nValTit := SE1->E1_SALDO
nMoeda := SE1->E1_MOEDA
aValores := FuncaMoeda(dDataBase,nValTit,nMoeda)
For i:=1 to 5
    ? "Valor do titulo na "+Str(i,1)+"| moeda: "+;
    Transform(aValores[i],"@E 9,999,999.99")
Next i
inkey(0)
Return
```

@... 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 através 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 especificada e na ordem especificada, 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 índice 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 Advanced. 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.

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 Advanced 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 RdMakes, as sintaxes descritas acima também são válidas, não devendo-se 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 Rdmake:

```
cCodNew := GetSx8Num(“SZ1”,“Z1_COD”)
// Processamento...
```

```
// Confirmacao  
ConfirmSx8()  
// ou Cancelamento  
RollBackSx8()  
Return
```

GravaOrcado

Tipo: Processamento e Planilha

Permite que um determinado valor calculado pela planilha seja gravado no Arquivo de Orçamentos.

Sintaxe

GravaOrcado(cConta,nCélula,nMês,nMoeda)

Parâmetros

- cConta – Conta Contábil a ser orçada
- nCélula – Número da célula onde o valor estará contido
- nMês – Mês a ser orçado (se nulo, será mês corrente)
- nMoeda – Moeda a ser orçada (se nula, será moeda nacional)

Exemplo

Para obter um valor referente à conta “11102001”, sendo que este deverá ser orçado na Contabilidade para o mês “07” e na moeda “1”. Para tanto, cria-se a seguinte expressão: GravaOrcado(“11102001”,#022,7,1)

Esta função irá devolver o conteúdo “<<< Orçado >>>”.

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
:
```

ImpCadast

Tipo: Impressão

Imprime relatório de cadastros padrões do SIGA Advanced. 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()

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
```

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 índice com a chave inversa, do maior valor para o menor.
- cFiltro – Expressão utilizada para filtro.
- cTexto – Texto da régua de processamento (“Selecionando registros ...”).

Retorno

Nil

Exemplo

```
DbSelectArea("SC5")
cFiltro := "C5_OK<>'X'"
cChave := "Dtos(C5_EMISSAO)+C5_VEND1"
cIndSC51 := CriaTrab(Nil,.F.)
IndRegua("SC5", cIndSC51, cChave, , cFiltro, "Selecionando Pedidos...")
```

LetterOrNum

Tipo: Processamento

Verifica se determinado caracter 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
```

MarkBrowse

Tipo: Processamento

Monta um browse padrão do sistema, permitindo marcar e desmarcar linhas.

Sintaxe

MarkBrowse(cAlias,cCampo,cCpo,aCampos,lMarc,cMarca,cCtrlM,lBotoes,cTopFun,cBotFun,aCoord)

Parâmetros

- cAlias – Álias 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)
```

MBrowse

Tipo: Processamento

Monta um browse padrão do sistema, conforme os parâmetros.

Sintaxe

mBrowse(nLinha1, nColuna1, nLinha2, nColuna2, cAlias, aFixe, cCpo, nPar, cCor, nOpc)

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

Exemplo

```
cCadastro := "Cadastro de Orcamentos"
aRotina   := {{ "Pesquisar", "AxPesqui", 0, 1}, ;
               { "Incluir", "ExecBlock('DEMOA', .F.)", 0, 3}, ;
               { "Alterar", "ExecBlock('DEMOB')", 0, 4}, ;
               { "Excluir", "ExecBlock('DEMOB', .F.)", 0, 5}}
```

```
MBrowse(6, 1, 22, 75, "SA1")
```

Media

Tipo: Processamento

Retorna a taxa média da moeda em um determinado mês/ano.

Sintaxe

Media(nMoeda,nMes,nAno)

Parâmetros

- nTaxa – Taxa média calculada.
- nMoeda – Moeda para cálculo da taxa média.
- nMes – Mês para cálculo da taxa média. Se não informado, é assumido o mês da data base.
- nAno – Ano para cálculo da taxa média. Se não informado, é assumido o ano da data base.

Exemplo

```
// Exemplo de uso da funcao Media
nTxMed := Media(1)
nTaxa   := nTxMed
@ 00,00 Say "Media do mes: "+Transform(nTxMed,"@E999.99")
@ 00,01 Say "Taxa calcul.: " Get nTaxa Picture "@E999.99"
Read
Return
```

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)
```

Modelo2

Tipo: Processamento

Exibe formulário para cadastro segundo o modelo 2 (como a rotina de Nota Fiscal).

Sintaxe

Modelo2(cTítulo,aCabec,aRodapé,aGd,nOp,cLOk,cTOk,
[aGetsGD,bF4,cIniCpos,nMax,aCordw,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 coordenadas 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

Exemplo

```
//*****  
// 3,4 Permite alterar getdados e incluir linhas  
// 6 So permite alterar getdados e nao incluir linhas  
// Qualquer outro numero so visualiza  
n0pcx:=3  
dbSelectArea("Sx3")  
dbSetOrder(1)  
dbSeek("SX5")  
nUsado:=0  
aHeader:={}  
While !Eof() .And. (x3_arquivo == "SX5")  
    IF X3USO(x3_usado) .AND. cNivel >= x3_nivel  
        nUsado:=nUsado+1  
        AADD(aHeader,{ TRIM(x3_titulo),x3_campo,;  
            x3_picture,x3_tamanho,x3_decimal,;  
            "ExecBlock('Md2valid',.f.,.f.)",x3_usado,;  
            x3_tipo, x3_arquivo, x3_context } )  
        Endif  
        dbSkip()  
    End  
    aCols:=Array(1,nUsado+1)  
    dbSelectArea("Sx3")  
    dbSeek("SX5")  
    nUsado:=0  
    While !Eof() .And. (x3_arquivo == "SX5")  
        IF X3USO(x3_usado) .AND. cNivel >= x3_nivel  
            nUsado:=nUsado+1  
            IF n0pcx == 3  
                IF x3_tipo == "C"  
                    aCOLS[1][nUsado] := SPACE(x3_tamanho)  
                ElseIf x3_tipo == "N"  
                    aCOLS[1][nUsado] := 0  
                ElseIf x3_tipo == "D"  
                    aCOLS[1][nUsado] := dDataBase  
                ElseIf x3_tipo == "M"  
                    aCOLS[1][nUsado] := ""  
            Else  
                aCOLS[1][nUsado] := .F.  
            Endif  
        Endif  
    Endif  
Endif
```

```

        Endif
dbSkip()
End
aCOLS[1][nUsado+1] := .F.
cCliente:=Space(6)
cLoja    :=Space(2)
dData    :=Date()
nLinGetD:=0
cTitulo:="TESTE DE MODELO2"
aC:={}
// aC[n,1] = Nome da Variavel Ex.: "cCliente"
// aC[n,2] = Array com coordenadas do Get [x,y], em
//           Windows estao 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.
#ifdef WINDOWS
    AADD(aC,{"cCliente" ,{15,10} ,"Cod. do Cliente","@"},;
    'ExecBlock("MD2VLCLI",.F.,.F.)',"SA1",})
    AADD(aC,{"cLoja"      ,{15,200},"Loja","@"},,,})
    AADD(aC,{"dData"      ,{27,10} ,"Data de Emissao",,,,})
#else
    AADD(aC,{"cCliente" ,{6,5} ,"Cod. do Cliente","@"},;
    ExecBlock("MD2VLCLI",.F.,.F.)',"SA1",})
    AADD(aC,{"cLoja"      ,{6,40},"Loja","@"},,,})
    AADD(aC,{"dData"      ,{7,5} ,"Data de Emissao",,,,})
#endif
aR:={}
// aR[n,1] = Nome da Variavel Ex.: "cCliente"
// aR[n,2] = Array com coordenadas do Get [x,y], em
//           Windows estao em PIXEL
// aR[n,3] = Titulo do Campo
// aR[n,4] = Picture
// aR[n,5] = Validacao
// aR[n,6] = F3
// aR[n,7] = Se campo e' editavel .t. se nao .f.
#ifdef WINDOWS
    AADD(aR,{"nLinGetD" ,{120,10},"Linha na GetDados",;
    "@E 999",,,,F.})
#else
    AADD(aR,{"nLinGetD" ,{19,05},"Linha na GetDados",;
    "@E 999",,,,F.})
#endif
#ifdef WINDOWS

```

```

        aCGD:={44,5,118,315}
#ELSE
        aCGD:={10,04,15,73}
#ENDIF
cLinhaOk := "ExecBlock('Md2LinOk',.f.,.f.)"
cTudoOk  := "ExecBlock('Md2TudOk',.f.,.f.)"
// lRet = .t. se confirmou
// lRet = .f. se cancelou
lRet:=Modelo2(cTitulo,aC,aR,aCGD,nOpcx,cLinhaOk,cTudoOk)
// No Windows existe a funcao de apoio CallMod20bj() que
// retorna o objeto Getdados Corrente
Return

```

Modelo3

Tipo: Processamento

Executa cadastro semelhante ao cadastro de Pedidos de Venda, cabeçalho variável com itens.

Sintaxe

Modelo3(cTitulo,cAliasEnchoice,cAliasGetD,aCpoEnchoice,cLinOk,cTudOk,nOpcE,nOpcG,cFieldOk,
[IVirtual,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)
- IVirtual – 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

- IRet – Retorno da função modelo3. Se True a operação foi confirmada.

Exemplo

```
aRotina := {{ "Pesquisa","AxPesqui", 0 , 1},;
             { "Visual","AxVisual", 0 , 2},;
             { "Inclui","AxInclui", 0 , 3},;
             { "Altera","AxAltera", 0 , 4, 20 },;
             { "Exclui","AxDeleta", 0 , 5, 21 }}
//ÚAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA;
//³ Opcoes de acesso para a Modelo 3
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAU
cOpcao:="VISUALIZAR"
Do Case
    Case cOpcao=="INCLUIR"; nOpcE:=3 ; nOpcG:=3
    Case cOpcao=="ALTERAR"; nOpcE:=3 ; nOpcG:=3
    Case cOpcao=="VISUALIZAR"; nOpcE:=2 ; nOpcG:=2
EndCase
//ÚAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA;
//³ Cria variaveis M->???? da Enchoice
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAU
RegToMemory("SC5",(cOpcao=="INCLUIR"))
//ÚAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA;
//³ Cria aHeader e aCols da GetDados
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAU
nUsado:=0
dbSelectArea("SX3")
dbSeek("SC6")
aHeader:={}
While !Eof().And.(x3_arquivo=="SC6")
    If Alltrim(x3_campo)="C6_ITEM"
        dbSkip()
    Loop
Endif
If X3US0(x3_usado).And.cNivel>=x3_nivel
nUsado:=nUsado+1
    Add(aHeader,{ TRIM(x3_titulo), x3_campo, x3_picture,;
                 x3_tamanho, x3_decimal,"AlwaysTrue()";;
                 x3_usado, x3_tipo, x3_arquivo, x3_context } )
Endif
dbSkip()
End

If cOpcao=="INCLUIR"
    aCols:={Array(nUsado+1)}
    aCols[1,nUsado+1]:=.F.
    For _ni:=1 to nUsado
        aCols[1,_ni]:=CriaVar(aHeader[_ni,2])
```

[illegible]

MontaF3

Tipo: Processamento

Permite o acesso à janela de Consultas Padronizadas (criadas no SXB) através 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
```

MovimCC

Tipo: Processamento

Retorna o movimento de um centro de custo mais conta contábil (extracontábil).

Sintaxe

MovimCC(cCC, cConta, nMês, nMoeda)

Parâmetros

- cCC – Código do centro de custo
- cConta – Código da conta contábil
- nMês – Referente ao mês
- nMoeda – Moeda desejada para obtenção do valor

Exemplo

MovimCC("3001", "111001", Month(Ddatabase), 1)

Movimento

Tipo: Processamento

Devolve o movimento (débito-crédito) de uma determinada conta, ou seja, o saldo do movimento.

Sintaxe

Movimento(cCódigo,nMês,nMoeda,dData)

Parâmetros

- cCódigo – Código da conta
- nMês – Mês do movimento desejado
- nMoeda – Moeda desejada para obtenção do movimento
- dData – Data do exercício desejado

Exemplo

Movimento("43",1,1, CTOD ("01/01/95")) - retorna o 1º período

Movimento(`43",1,1,CTOD("01/01/96")) - retorna o 13º período

```
// Exibe uma mensagem no dialogo
```

```
MsProcTxt("Processando produto: "+B1_COD)
```

```
// Processamento...
```

```
dbSkip()
```

```
EndDo
```

```
Return
```

MsGetVersion

Tipo: Processamento

Retorna array com as versões do Windows NT.

Sintaxe

MsGetVersion()

Exemplo

```
aVersao := MsGetVersion()
```

```
@ 001, 010 PSAY 'Versao do Windows : ' + Str( aVersao[1], 2 ) + '.' + Str(
aVersao[2], 2 )
```

```
@ 008, 010 PSAY 'Versao do Dos      : ' + Str( aVersao[3], 2 ) + '.' + Str(
aVersao[4], 2 )
```

MsgBox

Tipo: Tela Windows

Abre uma caixa de dialogo padronizada para informar o 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.

@..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.

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)

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
```

Orcado

Tipo: Processamento

Retorna o valor orçado de uma conta.

Sintaxe

Orcado(cConta, nPeríodo, nMoeda, dData)

Parâmetros

- cConta – Código da conta De
- nPeríodo – Referente ao período
- nMoeda – Moeda desejada para obtenção do valor
- dData – Data para conversão (em formato data ou character), caso não informada, será utilizada a DataBase do sistema

Exemplo

Orcado("11102001", 1, 1)

OrcadoCC

Tipo: Processamento

Recupera o valor orçado da Conta x Centro de Custo para utilização na planilha.

Sintaxe

OrcadoCC(cConta,cCC,nPeríodo,nMoeda,dData)

Parâmetros

- cConta – Código da Conta
- cCC – Código do Centro de Custo
- nPeríodo – Período (default mês da database)
- nMoeda – Moeda (default 1)
- dData – Data para conversão se moeda de 2 a 5 - (default dDataBase)

Exemplo

OrcadoCC("111001", "3001", 3, 2)

OpenFile

Tipo: Processamento

É a função que exibe o diagnóstico de arquivos, verificando a existência dos arquivos de dados e os índices do sistema, criando caso não existam, etc. Abre os arquivos de acordo com o módulo 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 através 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
```

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
```

Periodo

Tipo: Processamento

Devolve o período contábil de acordo com o exercício atual.

Sintaxe

Periodo(dData,nMoeda)

Parametros

- dData – Data a ser considerada. Se não for especificada, é assumida a data base.
- nMoeda – Moeda a ser considerada. Se não for especificada, é assumida a primeira moeda.

Retorna

- nPer – Período contábil retornado.

Exemplo

Veja o exemplo da função CALCSALDO.

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)

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)

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 RdMake Windows a ProcRegua só utiliza o primeiro parâmetro. No RdMake 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 próximo número
                        // sequencial
Return
```

@...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 se for efetuar uma atualização no mesmo, 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_EMITTASO With dEmissao,;
F1_LOJA With cLoja ,F1_FORNECE With Subs(cA100For,1,6)
dbUnLock()
```



Após a atualização do registro, deve-se executar a função **MsUnlock()**.

RecMoeda

Tipo: Processamento

Rotina para obtenção do valor da moeda desejada em determinada data.

Sintaxe

RecMoeda(dData,nMoeda)

Parâmetros

- dData – Data para obtenção do valor da moeda.
- nMoeda – Moeda desejada.

Retorna

nVMoeda – Retorna o Valor da moeda na data desejada.

Exemplo

```
// Exemplo de uso da funcao RecMoeda:  
nDolar := RecMoeda(dDataBase,2)  
nValDolar := SE1->E1_VALOR * nDolar  
Return
```


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()

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 character 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 SINDEIX

Comentários

Baseado no SINDEIX, abre todos os índices padrões para o arquivo em pauta.

Exemplo

```
Select SA1  
Index on A1_ACUM to TRAB  
:  
SINDEIX := RetIndex ("SA1")
```

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.

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”) })

Saldo

Tipo: Processamento

Calcula o saldo de uma determinada conta até o período informado e na moeda especificada.

Sintaxe

Saldo(cConta,nPer,nMoeda)

Parâmetros

- cConta – Código da conta desejada.
- nPer – Período contábil até o qual será feito o cálculo.
- nMoeda – Moeda desejada.

Retorna

- nSld – Retorna o Saldo da conta até o período desejado.

Exemplo

```
// Exemplo de uso da funcao Saldo:
cConta := SA1->A1_CONTA
nSl     := Saldo(cConta,Periodo(dDataBase,2),2)
cSl     := Transform(nSl,"@E 9,999,999.99")
Alert("O saldo da conta "+cConta+" na moeda 2 e': "+cSl)
Return
```

SaldoCC

Tipo: Processamento

Calcula o saldo atual em um determinado centro de custo ou conta (extracontábil). O arquivo SI3 deve estar posicionado para a obtenção dos valores para o cálculo.

Sintaxe

SaldoCC(cCC,cConta,nPer,nMoeda)

Parâmetros

- cCC – Centro de Custo desejado
- cConta – Código da conta
- nPer – Período até o qual o acúmulo deve ser calculado
- nMoeda – Moeda desejada

Retorna

- nSld – Saldo atual retorna do.

Exemplo

```
// Exemplo de uso da funcao SaldoCC:  
// Assumindo-se que o SI3 esta posicionado, nao e neces-  
// sario informar o centro de custo e a conta.  
nSld := SaldoCC(,12,1)  
return
```

SaldoCusto

Tipo: Processamento

Calcula o saldo dos centro de custos extracontábeis.

Sintaxe

SaldoCusto(cCC1,cCC2,cConta1,cConta2,nMês,nMoeda)

Parâmetros

- cCC1 – Centro de Custo Inicial
- cCC2 – Centro de Custo Final
- cConta1 – Conta Inicial
- cConta2 – Conta Final
- nMês – Mês (se nula, assume o mês de referência da database)
- nMoeda – Moeda (se nula, será assumido 1)

Exemplo

SaldoCusto("1007", "1099", "3", "4", "10, 1)

SaldoSB2

Tipo: Processamento

Esta função calcula o saldo atual do produto (do Arquivo SB2), descontando os valores empenhados, reservados, etc. É necessário que o Arquivo SB2 esteja posicionado no produto desejado.

Sintaxe

SaldoSB2()

Retorna

nSld – Retorna o Saldo do produto calculado.

Exemplo

```
// Exemplo de uso da funcao SaldoSb2:
cProd := Space(15)
@ 10,10 Get cProd Picture "@!"
Read
dbSelectArea("SB2")
dbSeek(cProd)
cSld := Transform(SaldoSb2(),"@E 9,999,999.99")
Alert("Este produto tem um saldo de: "+cSld)
Return
```

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 N° 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 )
```

SetPrint

Tipo: Impressão

Altera os padrões de impressão.

Sintaxe

SetPrint(cAlias,cNomeRel,cPerg,cDesc1,cDesc2,cDesc3,cDesc4,IDic, 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
- IDic – 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:= "C"
// 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
```

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.

SldBco

Tipo: Processamento

Retorna o saldo bancário em uma data.

Sintaxe

SldBco(cBanco,cAgência,cConta,dData,nMoeda)

Parâmetros

cBanco – Código do Banco

cAgência – Agência Bancária

cConta – Conta Bancária

dData – Data do Saldo

nMoeda – Moeda do Saldo Bancário

Exemplo

SldBco("409","00198","011122", dDataBase,1)

SldCliente

Tipo: Processamento

Retorna o saldo a receber do cliente em uma determinada data.

Sintaxe

SldCliente(cCliente,dData,nMoeda,lSaldo)

Parâmetros

- cCliente – Código do Cliente+Loja
- dData – Data do Movimento a Receber (padrão é dDataBase)
- nMoeda – Moeda
- lSaldo – Se .T. considera o saldo do SE5 (padrão é .T.)

Exemplo

SldCliente("00000101",dDataBase)

SldFornece

Tipo: Processamento

Retorna o saldo a pagar do fornecedor em uma data.

Sintaxe

SldFornece(cFornece,dData,nMoeda,lSaldo)

Parâmetros

- cFornece – Código do Fornecedor+Loja
- dData – Data do Movimento a Pagar (padrão é dDataBase)
- nMoeda – Moeda - (padrão é 1)
- lSaldo – Se .T. considera o saldo do SE5 (padrão é .T.)

Exemplo

SldFornece("00000101")

SldPagar

Tipo: Processamento

Retorna o saldo a pagar em uma determinada data.

Sintaxe

SldPagar(dData,nMoeda,lData)

Parâmetros

- dData – Data do Movimento a Pagar (padrão é dDataBase)
- nMoeda – Moeda (padrão é 1)
- lData – Se .T. Até a Data, .F. Somente Data (padrão é .T.)

Exemplo

SldPagar(dDataBase,1,.T.)

SldReceber

Tipo: Processamento

Retorna o saldo a receber em uma data.

Sintaxe

SldReceber(dData,nMoeda,lData)

Parâmetros

- dData – Data do Movimento a Receber.
- nMoeda – Moeda - default 1
- lData – .T. - até a Data; .F. - somente Data (o padrão é .T.)

Exemplo

SldReceber(Data,1,.T.)

SomaContas

Tipo: Processamento

Retorna o saldo acumulado de um grupo de contas, de acordo com a sintaxe apresentada. Esta função considera somente contas de classe “A” – analítica

Sintaxe

SomaContas(cLista,nMês,nMoeda)

Parâmetros

- cLista – Lista de contas, cercada por aspas (“”).
O separador “:” (dois pontos) informa intervalo de contas De-Até.
O separador “,” (vírgula) informa separação de contas.
- nMês – Mês (default mês da database)
- nMoeda – Moeda (default 1)

Exemplo

SomaContas(“11101001”,3,1)

Devolve o saldo da conta em questão no mês 3 na moeda 1.

SomaMovim

Tipo: Processamento

Retorna o movimento dentro de um intervalo de contas analíticas.

Sintaxe

SomaMovim(cConta1, cConta2, nMês, nMoeda)

Parâmetros

cConta1 – Código da conta De

cConta2 – Código da conta Até

nMês – Referente ao mês

nMoeda – Moeda desejada para obtenção do valor

Exemplo

SomaMovim("41304", "41305", 12, 1)

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.”.

SomaSaldo

Tipo: Processamento

Retorna o saldo atual entre um intervalo de contas.

Sintaxe

SomaSaldo(cConta1, cConta2, nPeríodo, nMoeda)

Parâmetros

cConta1 – Código da conta De

cConta2 – Código da conta Até

nPeríodo – Referente ao período

nMoeda – Moeda desejada para obtenção do valor

Exemplo

SomaSaldo("31001", "31010", 12, 1)

SumMovimCC

Tipo: Processamento

Retorna o movimento de um intervalo de centro de custos extracontábeis. Poderá ser parametrizados também um grupo de contas.

Sintaxe

SumMovimCC(cCC1,cCC2,cConta1,cConta2,nMês,nMoeda)

Parâmetros

- cCC1 – do Centro de Custo
- cCC2 – até Centro de Custo
- cConta1 – da Conta
- cConta2 – até a Conta
- nMês – Mês (default mês da database)
- nMoeda – Moeda (default 1)

Exemplo

SumMovimCC(“3001”, “3100”, “31001”, “31010”, 12, 1)

Tabela

Tipo: Processamento

Devolve o conteúdo da tabela de acordo com a chave. Esta função é usada para a obtencao do conteúdo de uma determinada tabela, na chave especificada. Retorna o conteudo, possibilitando inclusive a exibição de um “help” caso a tabela não exista.

Sintaxe

Tabela(cTab,cChav,IPrint)

Parâmetros

- cTab – Número da tabela a pesquisar (deve ser informado como caracter).
- cChav – Chave a pesquisar na tabela informada.
- IPrint – Indica se deve (.T.) ou não (.F.) exibir o help ou a chave NOTAB se a tabela não existir.

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 Character.

Sintaxe

Texto(ExpC)

Parâmetros

ExpC1 – Expressão a ser verificada

Exemplo

Texto()

@ ...T0

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 – Título 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

Variação

Tipo: Processamento

Retorna a variação em percentual entre dois valores.

Sintaxe

Variação(nFator1,nFator2)

Parâmetros

nFator1 – Primeiro fator comparativo

nFator2 – Segundo fator comparativo

Exemplo

Variacao(100000,50000)

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 é claro se o arquivo está compartilhado ou não (definição que é feita através do Módulo 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
```

XMoeda

Tipo: Processamento

Rotina para conversão de valores entre moedas.

Sintaxe

xMoeda(nVMo,nMo,nMd,dData,nDec)

Parâmetros

- nVMo – Valor na moeda origem.
- nMo – Número da moeda origem.
- nMd – Número da moeda destino.
- dData – Data para conversão.
- nDec – Número de decimais. Se não informado, assume-se 2 casas decimais.

Retorna

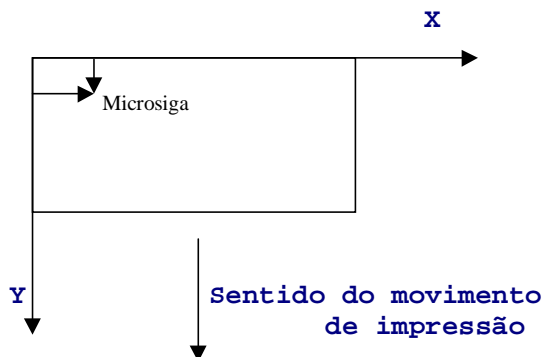
nVMoeda – Retorna o Valor na moeda de destino.

Exemplo

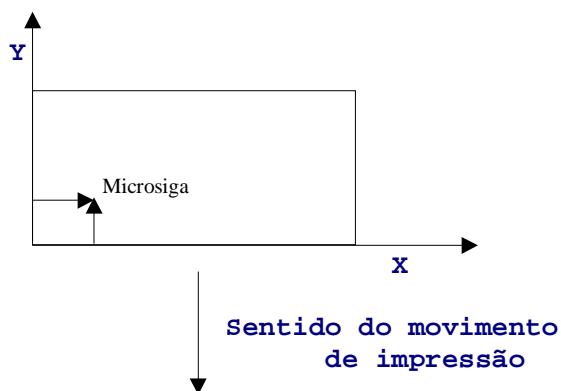
```
// Exemplo de uso da funcao xMoeda:  
nVal := SE1->E1_VALOR // Valor na moeda 1  
nVM3 := 0              // Contera o valor na moeda 3  
nVM3 := xMoeda(nVal,1,3,dDataBase)  
Return
```

Funções para impressão de etiquetas padrão ZPL e Allegro

Visualização do posicionamento da imagem na etiqueta no padrão Zebra (ZPL):



Visualização do posicionamento da imagem na etiqueta no padrão Allegro:



MSCBPRINTER

Tipo: Impressão

Configura modelo da impressora, saída utilizada, resolução na impressão e tamanho da etiqueta a ser impresso.

Sintaxe

MscbPrinter(cModelPrt, cPorta, nDensidade, nTamanho)

Parâmetros

- [ModelPrt] – String com o modelo de impressora Zebra
Os modelos disponíveis padrão Zebra:
S500-6, Z105S-6, Z16S-6, S300, S500-8, Z105S-8,
Z160S-8 ,Z140XI,Z90XI e Z170ZI.
Os modelos disponíveis padrão Allegro:
ALLEGRO, PRODIGY, DMX e DESTINY.
- cPorta – String com a porta
- [nDensidade] – Número com a densidade referente a quantidade de pixel por mm
- [nTamanho] – Tamanho da etiqueta em milímetros.



O parâmetro nDensidade não é necessário informar, pois ModelPrt o atualizará automaticamente. A utilização deste parâmetro (nDensidade) deverá ser feita quando não souber o modelo da impressora.

O tamanho da etiqueta será necessário quando a mesma não for continua.

Exemplo

MSCBPRINTER("S500-8", "COM2:9600,e,7,2",NIL, 42)

MSCBBEGIN

Tipo: Impressão

Inicializa a montagem da imagem para cada etiqueta

Sintaxe

MscbBegin(nQtde, nVeloc)

Parâmetros

- nQtde – Quantidade de cópias
- nVeloc – Velocidade (1,2,3,4,5,6) polegada por segundo

Exemplo

MSCBBEGIN(3,4)

MSCBEND

Tipo: Impressão

Finaliza a montagem da imagem.

Sintaxe

MscBend()

Parâmetros

Nil

Exemplo

```
MSCBEND()
```


MSCBSAY

Tipo: Impressão

Imprime um String.

Sintaxe

MscbSay(nXmm, nYmm, cTexto, cRotacao, cFonte, cTam, [lReverso],
[lSerial], [cIncr], [lZerorsL])

Parâmetros

- nXmm – Posição X em Milímetros
- nYmm – Posição Y em Milímetros
- cTexto – String a ser impressa
- cRotação – String com o tipo de Rotação (N,R,I,B)
 - N-Normal
 - R-Cima p/Baixo
 - I-Invertido
 - B-Baixo p/ Cima
- cFonte – String com o tipo de Fonte
 - Zebra : (A,B,C,D,E,F,G,H,0) 0(Zero) - fonte escalar
 - Alegro : (0,1,2,3,4,5,6,7,8,9) 9 - fonte escalar
- cTam – String com o tamanho da Fonte
- [lReverso] – Imprime em reverso quando tiver sobre um box preto
- [lSerial] – Serializa o código
- [cIncr] – String, valor de incremento da serie
- [lZerorsL] – Coloca zeros a esquerda na serie

Exemplo

MSCBSAY(15,3,"MICROSIGA ","N","C","018,010",.t.)

MSCBSAYBAR

Tipo: Impressão

Imprime Código de Barras.

Sintaxe

MscbSayBar(nXmm, nYmm, cConteudo, cRotacao, cTypePrt, nAltura, lIdigVer, lLinha, lLinBaixo, cSubSetIni, nLargura, nRelacao, lCompacta, lSerial, clncr)

Parâmetros

- | | |
|-----------|---|
| nXmm | – Posição X em Milímetros |
| nYmm | – Posição Y em Milímetros |
| cConteudo | – String a ser impressa |
| cRotação | – String com o tipo de Rotação |
| cTypePrt | – String com o Modelo de Código de Barras |
- Zebra
- 2 - Interleaved 2 of 5
 - 3 - Code 39
 - 8 - EAN 8
 - E - EAN 13
 - U - UPC A
 - 9 - UPC E
 - C - CODE 128
- Zebra
- D - Interleaved 2 of 5
 - A - Code 39
 - G - EAN 8
 - F - EAN 13
 - B - UPC A
 - C - UPC E
 - E - CODE 128
- | | |
|------------|--|
| [nAltura] | – Altura do código de Barras em Milímetros |
| [lIdigver] | – Imprime dígito de verificação |

[lLinha]	– Imprime a linha de código
[lLinBaixo]	– Imprime a linha de código acima das barras
[cSubSetIni]	– Utilizado no code 128
[nLargura]	– Largura da barra mais fina em pontos default 3
[nRelacao]	– Relação entre as barras finas e grossas em pontos default 2
[lCompacta]	– Compacta o código de barra
[lSerial]	– Serializa o código
[cIncr]	– String, valor de incremento da série
[lZerosL]	– Coloca zeros à esquerda na série

Exemplo

MSCBSAYBAR(20,22,AllTrim(SB1->B1_CODBAR),"N","C",13)

MSCBSAYMEMO

Tipo: Impressão

Monta e imprime um campo MEMO.

Sintaxe

MscbSayMemo(nXmm, nYmm, cConteudo, cRotacao, cTypePrt, nAltura, ldigVer, lLinha, lLinBaixo, cSubSetIni, nLargura, nRelacao, lCompacta, lSerial, cIncr)

Parâmetros

nXmm	– Posição X em Milímetros
nYmm	– Posição Y em Milímetros
nLMemomm	– Tamanho da 1 linha do campo memo em Milímetros
nQLinhas	– Quantidade de linhas
cTexto	– String a ser impressa
cRotação	– String com o tipo de Rotação (N,R,I,B)
cFonte	– String com o tipo de Fonte (A,B,C,D,E,F,G,H,0)
cTam	– String com o tamanho da Fonte
[lReverso]	– lógica se imprime em reverso quando tiver sobre um box preto
[cAlign]	– String com o tipo de Alinhamento do memo
	L - Esquerda
	R - Direita
	C - Centro
	J - Margem a margem (justificado)

Exemplo

MSCBSAYMEMO(1,10,8,4,cTexto,"N","A","9,5",.f., "C")

MSCBBOX

Tipo: Impressão

Imprime um Box.

Sintaxe

MscbBox(nX1mm, nY1mm, nX2mm, nY2mm, [nExpassura], [cCor])

Parâmetros

- nX1mm – Posição X1 em Milímetros
- nY1mm – Posição Y1 em Milímetros
- nX2mm – Posição X2 em Milímetros
- nY2mm – Posição Y2 em Milímetros
- [nExpassura] – Número com a espessura em pixel
- [cCor] – String com a Cor Branco ou Preto (“W” ou “B”)

Exemplo

MSCBBOX(12,01,31,10,37)

MSCBLineH

Tipo: Impressão

Imprime uma linha horizontal

Sintaxe

MscbLineH(nX1mm, nY1mm, nX2mm, [nExpassura], [cCor])

Parâmetros

- nX1mm – Posição X1 em Milímetros
- nY1mm – Posição Y1 em Milímetros
- nX2mm – Posição X2 em Milímetros
- [nExpassura] – Número com a espessura em pixel
- [cCor] – String com a Cor Branco ou Preto (“W” ou “B”)

Exemplo

MSCBLineH(01,10,80,3,“B”)

MSCBLineV

Tipo: Impressão

Imprime uma linha vertical.

Sintaxe

MscbLineV(nX1mm, nY1mm, nY2mm, [nExpassura], [cCor])

Parâmetros

- nX1mm – Posição X1 em Milímetros
- nY1mm – Posição Y1 em Milímetros
- nY2mm – Posição X2 em Milímetros
- [nExpassura] – Número com a espessura em pixel
- [cCor] – String com a Cor Branco ou Preto (“W” ou “B”)

Exemplo

MSCBLineV(01,10,80,3,“B”)

MSCBLOADGRF

Tipo: Impressão

Carrega uma imagem para memória da impressora.



Para o padrão Zebra, o arquivo do gráfico tem que ser do tipo GRF, gerado através de um PCX ou TIF no software fornecido pelo fabricante da zebra. Para o padrão Allegro, o arquivo do gráfico pode ser do tipo BMP, PCX eIMG. Não precisa ser convertido.

Sintaxe

MscbLoadGrf(cImagem)

Parâmetros

cImagem – nome do arquivo que será carregado

Exemplo

MSCBLOADGRF("LOGO.GRF")

MSCBGRAFIC

Tipo: Impressão

Imprime gráfico que está armazenado na memória da impressora.

Sintaxe

MscbGrafic(nXmm, nYmm, cArquivo, [lReverso])

Parâmetros

- nXmm – Posição X em Milímetros
- nYmm – Posição Y em Milímetros
- cArquivo – Nome do gráfico que foi carregado na memória da impressora (não colocar a extensão .GRF)
- [lReverso] – Imprime em reverso quando tiver sobre um box preto

Exemplo

MSCBGRAFIC(3,3,"LOGO",.t.)



Parâmetros que estiverem entre [], significa que não são OBRIGATÓRIOS.

MSCBWRITE

Tipo: Impressão

Permite enviar para porta uma linha de programação nativa da Impressora.

Sintaxe

MSCBWRITE(cConteúdo)

Parâmetros

cConteúdo – Linha de programa nativa da impressora.

Exemplo

```
MSCBWRITE("^F01,1^GB400,50,25^FS")
```



Parâmetros que estiverem entre [], significa que não são OBRIGATÓRIOS e os parâmetros que estiverem com *, significa que são ignorados no padrão Allegro.

Tipos de Fontes para Zebra:

As fontes A,B,C,D,E,F,G,H, são do tipo BITMAPPEDs, tem tamanhos definidos e podem ser expandidos proporcionalmente a dimensão mínima.

Exemplo:

Fonte do tipo A, 9 X 5 ou 18 x 10 ou 27 X 15 ...

A fonte 0 (Zero) é do tipo ESCALAR, esta será gerada na memória da impressora, portanto, torna-se um processamento lento.

<div>Fonte A 9x5</div> <div>Fonte A 18x10</div> <div>Fonte A 27x15</div> <div>Fonte A 36x20</div>	<div>Fonte F 26x13</div> <div>Fonte F 52x26</div> <div>Fon F 78x39</div>
<div>Fonte B 12x7</div> <div>Fonte B 22x14</div> <div>Fonte B 33x21</div> <div>Fonte B 44x29</div>	<div>Fonte G 60x40</div>
<div>Fonte C 18x10</div> <div>Fonte C 36x20</div> <div>Fonte C 54x30</div> <div>Fonte C 72x29</div>	<div>Fonte H 21x13</div> <div>Fonte H 42x26</div> <div>FON H 84x39</div>
<div>Fonte D 18x10</div> <div>Fonte D 36x20</div> <div>Fonte D 54x30</div> <div>Fonte D 72x29</div>	<div>Fonte E 28x15</div> <div>Fonte E 56x30</div> <div>Fon E 84x45</div>
	<div>Fonte F 26x13</div> <div>Fonte F 52x26</div> <div>Fonte F 78x39</div> <div>Fonte G 60x40</div> <div>Fonte G 120x80</div> <div>Fonte H 21x13</div> <div>Fonte H 42x26</div> <div>Fonte H 84x39</div> <div>Fonte I 42x26</div> <div>Fonte I 84x39</div> <div>Fonte J 42x26</div> <div>Fonte J 84x39</div> <div>Fonte K 42x26</div> <div>Fonte K 84x39</div> <div>Fonte L 42x26</div> <div>Fonte L 84x39</div> <div>Fonte M 42x26</div> <div>Fonte M 84x39</div> <div>Fonte N 42x26</div> <div>Fonte N 84x39</div> <div>Fonte O 40x40</div> <div>Fonte O 35x50</div>

Tipos de Fontes para Allegro:

As fontes 0,1,2,3,4,5,6,7,8, são do tipo BITMAPPEDs, tem tamanhos definidos e podem ser expandidos.

Exemplo:

Fonte do tipo 1, 1 X 1 ou 1 x 2 ou 1 X 3 ou 2 X 1 ou 2 X 2 ...

A fonte 9 (Nove) é do tipo ESCALAR, esta será gerada na memória da impressora, portanto, torna-se um processamento lento, as dimensões deste tipo de fonte tem que ser passado junto com o tamanho da fonte.

Exemplo: cTam := "001,001,002"

Dimensão da fonte que pode ser de 0 a 9
Tamanho da fonte

FONT 0 1x1 FONT 0 1x2 FONT 0 1x3 FONT 0 2x1 FONT 0 2x2	FONT 9 1x1x1 FONT 9 1x1x2 FONT 9 1x1x3 FONT 9 1x1x4 FONT 9 1x1x5 FONT 9 1x1x6 FONT 9 1x1x7 FONT 9 1x1x8
FONT 1 1x1 FONT 1 1x2 FONT 1 1x3 FONT 1 2x1 FONT 1 2x2	FONT 9 1x2x1 FONT 9 1x2x2 FONT 9 1x2x3 FONT 9 1x2x4
FONT 2 1x1 FONT 2 1x2 FONT 2 1x3 FONT 2 2x1 FONT 2 2x2	FONT 9 1x3x1 FONT 9 1x3x2 FONT 9 1x3x3 FONT 9 1x3x4
FONT 3 1x1 FONT 3 1x2 FONT 3 1x3 FONT 3 2x1 FONT 3 2x2	FONT 9 2x1x1 FONT 9 2x1x2 FONT 9 2x1x3 FONT 9 2x1x4
FONT 4 1x1 FONT 4 1x2 FONT 4 2x1 FONT 4 2x2	FONT 9 2x2x1 FONT 9 2x2x2 FONT 9 2x2x3 FONT 9 2x2x4

Utilizando as funções:

Exemplo: padrão Zebra

```
MSCBPRINTER("S500-8","COM2:9600,e,7,2",,42) //Seta tipo de impressora
padrao ZPL
MSCBLOADGRF("LOGO.GRF") //Carrega o logotipo para impressora
MSCBBEGIN(2,4) //Inicio da Imagem da Etiqueta
//com 2 copias e velocidade 4 etiquetas por polegadas
MSCBBBOX(01,01,80,40) //Monta BOX
MSCBBBOX(12,01,31.5,10,37)
MSCBGRAFIC(2.3,2.5,"LOGO") //Posiciona o logotio
MSCBSAY(15,3,"MICROSIGA ","N","C","018,010") //Imprime Texto
MSCBSAY(13,6.5,"SOFTWARE S/A","N","C","018,010")
MSCBLineH(01,10,80) //Monta Linha Horizontal
MSCBSAY(35,2,"Código Interno","N","B","11,7")
MSCBSAY(35,5,SB1->B1_COD,"N","E","28,15")
MSCBSAY(4,12,"Descricao","N","B","11,7")
MSCBSAY(4,16,SB1->B1_DESC,"N","F","26,13")
MSCBLINEH(01,20,80)
MSCBSAYBAR(20,22,AllTrim(SB1->B1_CODBAR),"N","C",13,.f.,.t.,,3,2,.t.)
//monta código de barras
MSCBEND() //Fim da Imagem da Etiqueta
```



Utilizando as funções:

Exemplo: padrão Allegro

```
.  
.   
.   
.   
  
MSCBPRINTER("S500-8","COM2:9600,e,7,2",,42) //Seta tipo de impressora  
padrao ZPL  
MSCBLOADGRF("LOGO.GRF") //Carrega o logotipo para impressora  
MSCBBEGIN(2,4) //Inicio da Imagem da Etiqueta  
    //com 2 copias e velocidade 4 etiquetas por polegadas  
    MSCBBOX(01,01,80,40)    //Monta BOX  
    MSCBBOX(12,01,31.5,10,37)  
    MSCBGRFIC(2.3,2.5,"LOGO")    //Posiciona o logotipo  
    MSCBSAY(15,3,"MICROSIGA ", "N", "C", "018,010") //Imprime Texto  
    MSCBSAY(13,6.5,"SOFTWARE S/A", "N", "C", "018,010")  
    MSCBLineH(01,10,80)    //Monta Linha Horizontal  
    MSCBSAY(35,2,"Código Interno", "N", "B", "11,7")  
    MSCBSAY(35,5,SB1->B1_COD, "N", "E", "28,15")  
    MSCBSAY(4,12,"Descricao", "N", "B", "11,7")  
    MSCBSAY(4,16,SB1->B1_DESC, "N", "F", "26,13")  
    MSCBLINEH(01,20,80)  
    MSCBSAYBAR(20,22,AllTrim(SB1->B1_CODBAR), "N", "C", 13,.f.,.t.,,3,2,.t.)  
    //monta código de barras  
MSCBEND() //Fim da Imagem da Etiqueta  
  
.   
.   
. 
```





RdMAKE EM AMBIENTE SQL

A princípio não existem diferenças na programação do *RDMAKE* na versão *SQL*, já que pelo próprio fato de ser uma linguagem interpretada, o sistema é quem se encarrega de executar os comandos e funções adequadamente no ambiente em que trabalha. Mas é importante saber algumas informações ao programar para o ambiente *SQL*. Deve-se lembrar que estamos trabalhando com um banco de dados relacional, que se utiliza de tabelas ao invés de arquivos, e onde o sistema não tem acesso aos dados de forma nativa e sim através do *Top Connect*. Essa forma de acesso adiciona ao sistema algumas das características e vantagens oferecidas pelo *SGBD* em uso (por exemplo, o *Oracle*, *MSSQL Server* ou o *DB2*) como, por exemplo, facilidades da linguagem *SQL*, mas por outro lado tem-se também as implicações da conversão dos comandos no padrão *xBase* para a perfeita compreensão no ambiente *SQL*.

Imagine a montagem de uma expressão de filtro para um índice condicional. Tome a seguinte expressão como exemplo: “**DTOS(E1_VENCTO) >= DTOS(mv_par01)**”. Em um ambiente padrão *xBase*, como o *NTX* ou o *ADS*, pode-se utilizar variáveis sem qualquer problema em uma expressão de filtro pois a mesma será avaliada registro a registro durante a montagem do índice. Mas no ambiente *SQL*, o filtro nada mais é do que uma tabela temporária, onde estão selecionados apenas os registros conforme a condição indicada. A seleção de dados em tabelas pelo *SQL* é mais rápida, mas em compensação o *SGBD* não tem como reconhecer a variável informada na expressão. Ela existe apenas no sistema ou, mais especificamente, no seu programa. Por isso, deve-se substituir a expressão anteriormente exemplificada pela seguinte (que também funcionaria perfeitamente em um ambiente *xBase*): “**DTOS(E1_VENCTO) >= “+DTOS(mv_par01)+”**”. Esta expressão é melhor que anterior simplesmente porque não se utiliza da variável e sim do conteúdo da mesma, o que pode ser compreendido em qualquer ambiente. Toda essas explicações são válidas, da mesma maneira, a filtros criados através do comando *SET FILTER*.

Ainda existem outros detalhes a se considerar quando se trabalha com índices em um ambiente *SQL*. É que na verdade não existem índices condicionais nesse ambiente. O filtro é criado independente do índice. Então, você pode criar um *INDREGUA* com um filtro e mudar a ordem, mas o filtro permanecerá ativo, em qualquer ordem. Do mesmo modo, não se pode manter dois índices, com filtros diferentes, pois um filtro sobrescreveria o outro.

Outro ponto de atenção deve ser a função *xBase* chamada *DBSETINDEX*. Podem ocorrer alguns erros ao tentar-se utilizar essa função para abrir um índice de trabalho criado. Por esses motivos e pelo fato de tornar o processamento mais lento deve-se evitar ao máximo o uso de índices de trabalho no ambiente *SQL*.

Da mesma maneira que a função *DBSETINDEX*, os comandos *COPY TO* e *APPEND FROM* também devem ter uma atenção especial. No ambiente *SQL* esses comandos são executados entre uma tabela e um arquivo *DBF* (e vice-versa) ou entre dois arquivos *DBF*. Por exemplo, o comando *COPY TO* pode ser usado para copiar os dados da tabela ativa para um *DBF* local e o comando *APPEND FROM* pode ser usado para importar os dados de um arquivo local para a tabela ativa. Os dois podem ser usados entre dois arquivos, mas nunca pode-se usar, por exemplo, o comando *APPEND FROM* para importar os dados de uma tabela para outra.

Apesar de todos esses detalhes, o ambiente *SQL* traz muitas facilidades ao interpretador *RDMAKE*. Através de novos comandos criados especificamente para esse ambiente, pode-se usar cláusulas *SQL* ou até mesmo *Stored Procedures* dentro de um *RDMAKE*. A seguir uma lista dos novos comandos e sintaxes (todos estes comandos podem ser consultados no programa de exemplo chamado *RDDEMO.PRW*, que acompanha o sistema na versão):

Comando ou Função	Descrição	Exemplo
TCQUERY	Executa uma query (consulta) ao banco de dados, retornando um alias para o resultado. Semelhante ao comando USE padrão do CLIPPER, porém executando a consulta. A sintaxe é a seguinte: TCQUERY <cQuery> [NEW] [ALIAS <cAlias>] Obs.: A tabela retornada é SEMPRE Read Only (somente leitura)	CQuery := "Select * From SA1990 For A1_COD > '200" TCQUERY cQuery NEW Alias "CNT" // Neste momento, alias CNT // disponível com o resultado // da consulta
TCSQLEXEC	Esta função executa qualquer sintaxe SQL (não somente consultas como a anterior). Pode ser usada para executar UPDATE, INSERT, DELETE, etc. A sintaxe é a seguinte: TCSQLEXEC(cClause) Obs.: A sintaxe de criação de uma Stored Procedure é dependente do SGBD em que o sistema trabalha. Ou seja, existem sintaxes específicas para o Oracle, para o MSSQL SERVER, para o SYBASE, etc.	// Cria uma Stored Procedure // chamada TSOMA CClause := "Create Procedure TSOMA (IN_V1 in number, OUT_RET out number) is BEGIN OUT_RET:=IN_V1+IN_V2"+CHR(59)+"END"+CHR(59) TCSQLEXEC(cClause)
TCSPEXEC	Esta função executa uma Stored Procedure. Aceita parâmetros e o retorno da função é sempre um array (com as variáveis de retorno). A sintaxe é a seguinte: aRet:=TCSPEXEC (cClause, [uPar1], [uPar2],...)	// Executa a Stored Procedure criada no // exemplo da função anterior aRet:= TCSPEXEC("TSOMA",2,3) ? aRet[1] // Imprime o resultado da // soma (no caso, 5)

EXECUTANDO

Simplesmente clique na opção do menu que o programa procurará no diretório corrente pelo programa contido no menu e se encontrado o executará, caso contrário, uma mensagem de aviso será gerada.

Executando com Debug



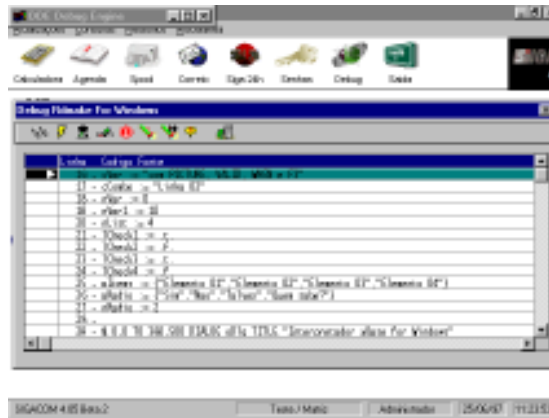
O botão na barra de ferramentas da janela principal permite a utilização do Debug para Rdmake. Este programa permite a visualização do arquivo fonte junto a execução, acesso à lista de variáveis, criação de pontos de parada, execução de qualquer comando independente ou mesmo fazer uma procura no programa fonte.

Para acessar o Debug, clique neste botão da barra de ferramentas. A janela para escolha do arquivo fonte a ser depurado será apresentada.






Após a seleção do arquivo PCODE (_IW), será apresentada a janela do Debug com o programa escolhido.


O código fonte (.PRW ou .PRX) deve estar no diretório de trabalho (\SIGADV).

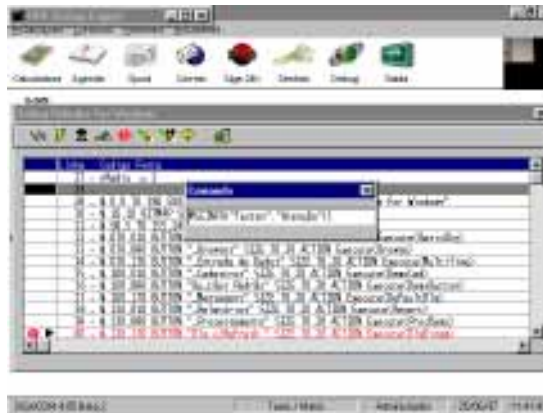



Utilize os botões ou as teclas para depuração:

-  - [F8] executa cada instrução do programa e retorna a janela do Debug.
-  - [F5] executa o programa até encontrar um ponto de parada definido pelo usuário.
-  - [F2] visualiza o conteúdo da variável informada.

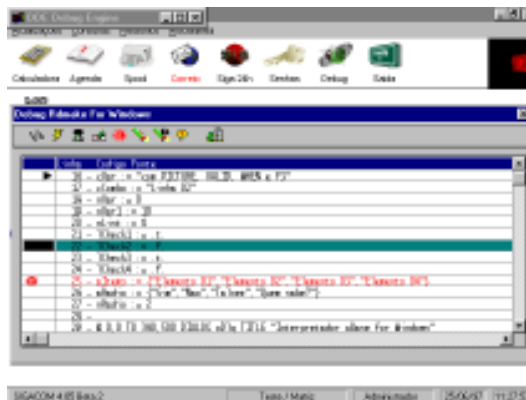


 - [F3] permite a execução de um comando informado.



 - [F9] define a linha posicionada como ponto de parada na execução do programa.

Para marcar ou desmarcar um ponto de parada, pode-se também dar um duplo clique na linha.





permite a localização de um texto no programa.



permite localizar a próxima ocorrência do texto.



acessa o help do Debug



finaliza o Debug.

Clique o botão direito do mouse ou pressione a tecla [F11] para apresentar um menu popup com as opções da barra de ferramentas.



O Debug para RdMake possui limitações no Windows 3.11, sendo necessário reinicializar o equipamento após o seu uso.

A estrutura de erros do RDMAKE obedece aos seguintes princípios:

(nnn) <Mensagem elucidativa do erro>

Linha do programa fonte onde ocorreu o erro

Erros e ações a serem tomadas

RDMAKE Fatal 3-> Erro Fatal na compilação da rotina TESTE Impossível continuar.”);

Causa - O ADVPL16 detectou algum erro de sintaxe

Ação - Corrigir o erro e resubmeter o programa ao RDMAKE

O termo LOCAL não é permitido.

Causa - Tentativa de declarar uma variável LOCAL

Ação - Retirar o termo LOCAL da declaração da variável

Não são permitidas funções STATIC.

Causa - Tentativa de declarar uma variável STATIC

Ação - Retirar o termo STATIC da declaração da variável

O termo PRIVATE não é permitido.

Causa - Tentativa de declarar uma variável PRIVATE

Ação - Retirar o termo PRIVATE da declaração da variável

Passagem de parâmetros não permitida. Use variáveis PRIVATE.

Causa - Na chamada de uma função interna ou na declaração de função foi solicitada a passagem de parâmetros

Ação - Ao invés de passar parâmetros para a função, declare a variável de retorno antes da chamada da função e mova um conteúdo para esta na função chamada.

Sintaxe inválida __<variável>.

Causa - Tentativa de criar uma variável ou função com __ (duplo underline) no início desta.

Ação - Variáveis deste tipo são prerrogativas do RDMAKE, Interpretador e do ADVPL16, declare variáveis com Caracteres Alfabéticos no início.

Operador de igualdade inválido, use :=, ==, x:=x+2, etc...

Causa - Uso de igualdade simples.

Ação - Use := para atribuir um valor a uma variável e == para comparar.

Linha maior que 512 bytes, truncada.

Causa - Uma linha maior que 512 caracteres foi inserida no programa fonte.

Ação - Reduza a linha.

Erro -> String muito grande.

Causa - Erro interno do RDMAKE.

Ação - Avise a Microsiga do problema.

*Uso de operadores inválidos ++ -- *= +=... use operadores xBase*

Causa - Uso de operadores compostos.

Ação - Ao invés de operadores compostos ex.: var++ use operadores simples
var:=var+1

EXEMPLOS DE PROGRAMA

/*

Função	RdDemo
Autor	Ary Medeiros
Data	15/02/96
Uso	RdMake <Programa.Ext> - w
Exemplo	RdMake RDDemo.prw -w

*/

```
cVar := "com PICTURE, VALID, WHEN e F3"  
cCombo := 2  
nVar := 0  
nVar1 := 10  
nList := 4  
ICheck1 := .t.  
ICheck2 := .f.  
ICheck3 := .t.  
ICheck4 := .f.  
aItems := {01,02,03,04}  
aRadio := {1,2,3,4}  
nRadio := 2
```

Define
Variáveis

Define um formulário "tela", a variável *oDlg* recebe todas as definições do objeto. Semelhante a criação de um objeto *TBrowseDb* / *TBrowseNew*.

```
@ 0,0 TO 380,500 DIALOG oDlg TITLE "Interpretador xBase for Windows"
```

```
@ 10,10 BITMAP SIZE 110,40 FILE "RDDemo.BMP"
```

Atribui uma figura *BitMap* ao objeto *oDlg*.

```
@ 60,5 TO 155,245
```

Atribui um *Box* simples ao objeto *oDlg*, semelhante ao comando *Box* (obsoleto) e a função *DispBox()* padrão do Clipper.

```
@ 070,010 BUTTON "_Objetos Básicos" SIZE 70,20 ACTION Execute(BasicObj)  
@ 070,090 BUTTON "_Browses" SIZE 70,20 ACTION Execute(Browse)  
@ 070,170 BUTTON "_Entrada de Dados" SIZE 70,20 ACTION Execute(Multiline)  
@ 100,010 BUTTON "_Cadastrros" SIZE 70,20 ACTION Execute(DemoCad)  
@ 100,090 BUTTON "Bo_tões Padrão" SIZE 70,20 ACTION Execute(DemoButton)  
@ 100,170 BUTTON "_Mensagens" SIZE 70,20 ACTION Execute(DefaultDlg)  
@ 130,010 BUTTON "_Relatórios" SIZE 70,20 ACTION Execute(Report)  
@ 130,090 BUTTON "_Processamento" SIZE 70,20 ACTION Execute(ProcDemo)  
@ 130,170 BUTTON "Dlg c/Refresh " SIZE 70,20 ACTION Execute(DlgDinam)  
@ 162,218 BMPBUTTON TYPE 1 ACTION Close(oDlg)
```

Atribui botões ao objeto *oDlg*. Quando o objeto estiver ativo estes botões executam as funções definidas na sua criação.

```
ACTIVATE DIALOG oDlg CENTERED
```

Para que todas as definições estejam disponíveis para uso, é necessário ativar o objeto *oDlg*. Esta função funciona de forma semelhante a definição de vários *Get's* e sua execução com o comando *Read*. Pode-se também comparar a definição dos objetos de um formulário ao Método *AddColumn*.

```
Return
```


/*

Função	BasicObj
Autor	Ary Medeiros
Data	15/02/96
Descrição	Objetos básicos Windows

*/

Function BasicObj

@ 0,0 TO 250,450 DIALOG oDlg1 TITLE "Objetos básicos do Windows"

@ 10,10 SAY "ListBox:"

Comando Say Padrão do Clipper.

@ 20,10 LISTBOX nList ITEMS altems SIZE 50,50

Este Comando atribui ao objeto *oDlg1* uma caixa de seleção **ListBox**, é semelhante a função Achoice() Padrão do Clipper.

@ 16,70 TO 67,120 TITLE "CheckBox"

Atribui ao objeto um *Box* que contem o Titulo especificado.

@ 23,75 CHECKBOX "Opção 1" VAR lCheck1

@ 33,75 CHECKBOX "Opção 2" VAR lCheck2

@ 43,75 CHECKBOX "Opção 3" VAR lCheck3

@ 53,75 CHECKBOX "Opção 4" VAR lCheck4

Este conjunto de Comandos atribui ao objeto *oDlg1* uma caixa de seleção **CheckBox**, a **CheckBox** cria uma lista de itens que podem ser marcados e desmarcados semelhante a criar uma serie de *Get's* para selecionar entre "S" ou "N". Não tem barras de rolagem como a **ListBox**.

@ 16,130 TO 67,180 TITLE "RadioButton"

Atribui ao objeto um *Box* que contém o Titulo especificado.

@ 23,130 RADIO aRadio VAR nRadio

Este Comando atribui ao objeto *oDlg1* uma caixa de seleção **RadioButton**, a **RadioButton** é semelhante a **CheckBox** a única diferença entre os comandos é que na **RadionButton** apenas um dos itens pode ser selecionado e **CheckBox** podem ser selecionados vários itens, a **CheckBox** utiliza variáveis Lógicas e a **RadioButton** utiliza uma Matriz. Não tem barras de rolagem como a **ListBox**.

@ 75,10 Say "ComboBox:"

Comando Say Padrão do Clipper.

@ 85,10 COMBOBOX cCombo ITEMS altems SIZE 50,50

Este Comando atribui ao objeto *oDlg1* uma caixa de seleção **ComboBox**, tem funcionamento semelhante a **ListBox** a única diferença entre os comandos é que na **ComboBox** os itens não são apresentados até que a caixa seja selecionada é ideal para pequenas áreas no formulário. Ambas utilizam uma Matriz..

@ 75,70 Say "Entrada de dados:"

Comando Say Padrão do Clipper.

@ 85,70 GET cVar PICTURE "@!" VALID .t. F3 "S11"

O comando *Get* no **RdMake For Windows** tem duas diferenças, a cláusula **F3**, que disponibiliza a consulta padrão conforme o parâmetro e não é necessário utilizar o comando *Read*.

@ 20,185 BUTTON "_Ok" SIZE 35,15 ACTION Close(oDlg1)

@ 45,189 BMPBUTTON TYPE 12 ACTION PlayWave("RDDemo.wav")

Definição de botões.

ACTIVATE DIALOG oDlg1 CENTER

Ativa o objeto *oDlg1*

Return

MANUAL RdMAKE - SIGA ADVANCED 4.07 - 201

/*

Função	DefaultDlg
Autor	Ary Medeiros
Data	15/02/96
Descrição	Diálogos Padronizados

*/

Function DefaultDlg

MsgBox ("Dialogo para mensagens de erro","Erro!!!","STOP")
MsgBox ("Informação ao usuário","Informação","INFO")
MsgBox ("Mensagens de alerta","Atenção","ALERT")
MsgBox ("Dialogo com perguntas...","Escolha","YESNO")
MsgBox ("mais perguntas...","Escolha","RETRYCANCEL")

A função MsgBox() pode ser utilizada para avisar o usuário de uma operação bem sucedida, um erro ou solicitar uma confirmação de processamento. O BitMap e os botões dependem do tipo selecionado nos parâmetros.

Return

/*

Função	Browse
Autor	Ary Medeiros
Data	15/02/96
Descrição	Objeto para manipulação de Browses

*/

Function Browse

@ 200,1 TO 400,530 DIALOG oDlg2 TITLE "Utilização de Browses"

Define Objeto oDlg2

@ 6,5 TO 93,150 BROWSE "SX6" ENABLE "!X6_CONTEUD"

Este comando atribui ao objeto oDlg2 o **Browse** padrão do sistema utilizando as definições do dicionário de dados. A cláusula ENABLE identifica o campo que se vazio alterna a “marca” entre as cores vermelho/verde.

@ 10,155 SAY "- Baseado no dicionário de dados"
@ 20,155 SAY "- Registros Habilitados/desabilitados"
@ 30,155 SAY "- Marcar/Desmarcar registros"
@ 40,155 SAY "- Movimentação/Resize de colunas"

Comando Say Padrão do Clipper.

@ 70,180 BUTTON "_Ok" SIZE 40,15 ACTION Close(oDlg2)

Definição de um botão.

ACTIVATE DIALOG oDlg2 CENTERED

Ativa o objeto oDlg2

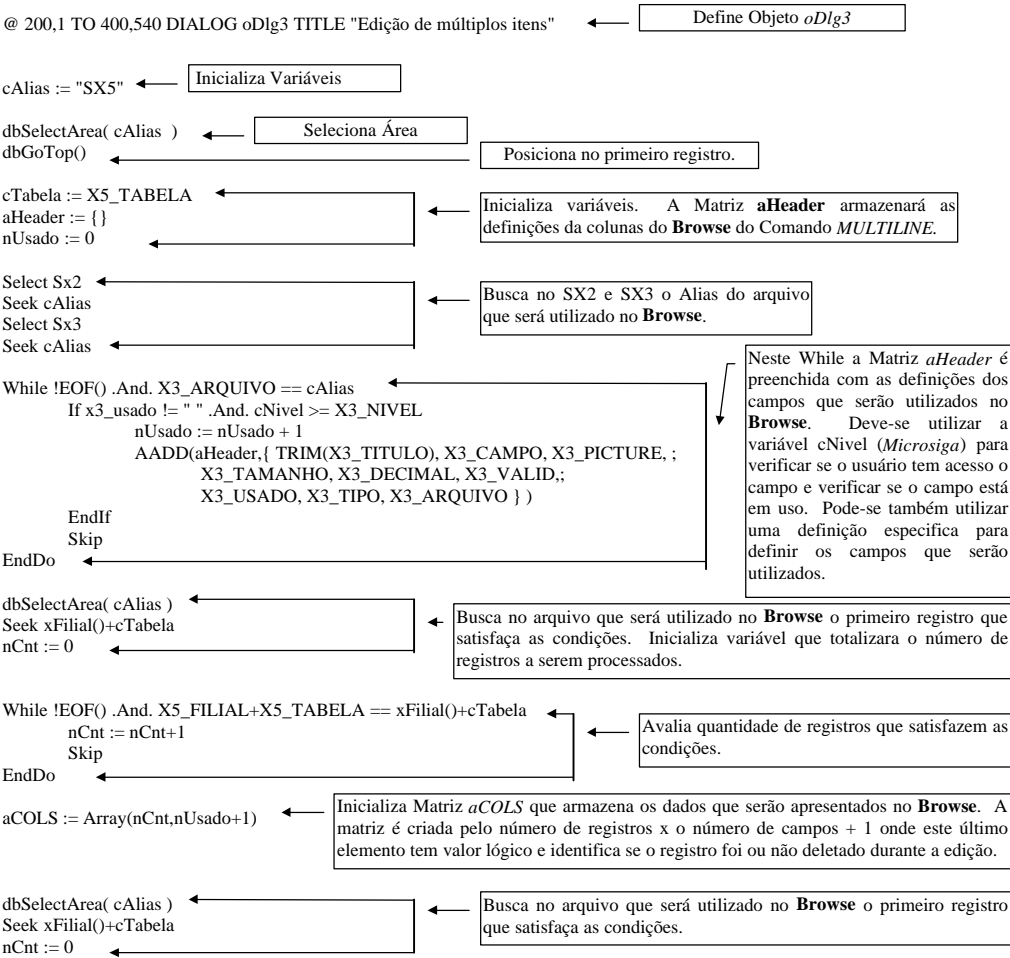
Return

```
/*
```

Função	MultiLine
Autor	Ary Medeiros
Data	15/02/96
Descrição	Objeto para entrada de dados(Itens)

```
*/
```

Function Multiline



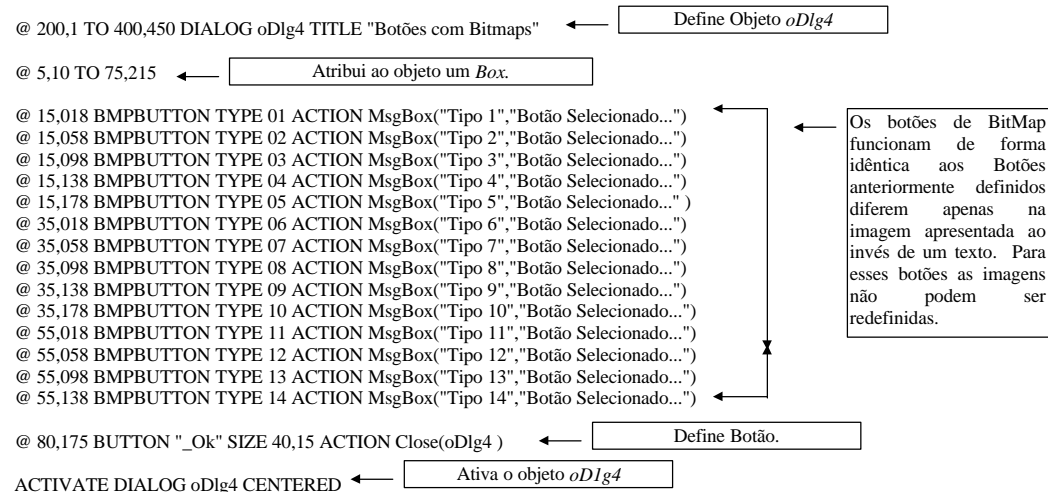
1

/*

Função	DemoButton
Autor	Ary Medeiros
Data	15/02/96
Descrição	Botões com BitMaps padronizados

*/

Function DemoButton



Return

/*

Função	DemoCad
Autor	Ary Medeiros
Data	15/02/96
Descrição	tela de cadastramento padrão

*/

Function DemoCad

AxCadastro("SX6","AX_Cadastro() - Cadastramento Padrão")
Return

A função AxCadastro() disponibiliza o Browse Padrão Microsiga e as rotinas padrão de Pesquisa, Visualiza, Altera, Inclui e Exclui. O padrão de edição da AxCadastro é o mesmo utilizados em cadastros como "Clientes" e "Produtos", Modelo 1.

Função	Report
Autor	Ary Medeiros
Data	15/02/96
Descrição	Exemplo de programação relatórios

Function Report

Define variáveis padrão em relatórios no SigaAdv.

```
m_pag := 0 //Variável que acumula numero da pagina
```

```
SetPrint(cString,wnrel,,titulo,cDesc1,cDesc2,cDesc3,.F.,"" ,tamanho)
```

A função SetPrint permite a configuração dos parâmetros do relatório

Endif

Verifica se foi pressionada tecla de interrupção, se verdadeiro abandona a função.

Assume as configurações definidas para impressão do relatório.

Endif

Verifica se foi pressionada a tecla de interrupção, se verdadeiro abandona a função.

No **RdMake** For Windows a definição de um relatório deve obedecer duas etapas , a configuração do relatório e sua impressão. Deve-se utilizar esta função para executar a função que fará a impressão do relatório.

Return

/*

Função	RptDetail
Autor	Ary Medeiros
Data	15/02/96
Descrição	Impressão do corpo do relatório.

*/

Function RptDetail

SetRegua(50) //Ajusta numero de elementos da régua de relatórios

A função SetRegua() inicializa a régua padrão de relatórios **RdMake** *Dos/Windows*.

Cabec(titulo,cabec1,cabec2,nomeprog,tamanho,18) //Impressao do cabeçalho

Função padrão para impressão do cabeçalho.

For nLin := 1 to 50
 if lEnd
 Exit
 endif
 @ nLin+8,0 PSAY "Linha ==>"
 @ nLin+8,10 PSAY nLin PICTURE "@E 999,999.99"
 IncRegua()
Next

Contador para demonstrar impressão. Deve-se utilizar PSAY no **RdMake** *For Windows*, quando for desenvolvido um relatório para *Dos/Windows* deve-se utilizar a Diretiva :
#IFDEF WINDOWS
#DEFINE PSAY SAY
#ENDIF
A Função IncRegua() incrementa a régua padrão.

Roda(0,"","P")

Função padrão para impressão do rodapé dos relatórios.

Set Filter To

Elimina filtro do arquivo principal.

If aReturn[5] == 1
 Set Printer To
 Commit
 ourspool(wnrel)
Endif

Testa se o relatório foi direcionado para disco , esvazia os Buffers e chama função de Spool Padrão.

MS_FLUSH()

Libera fila de relatórios em spool.

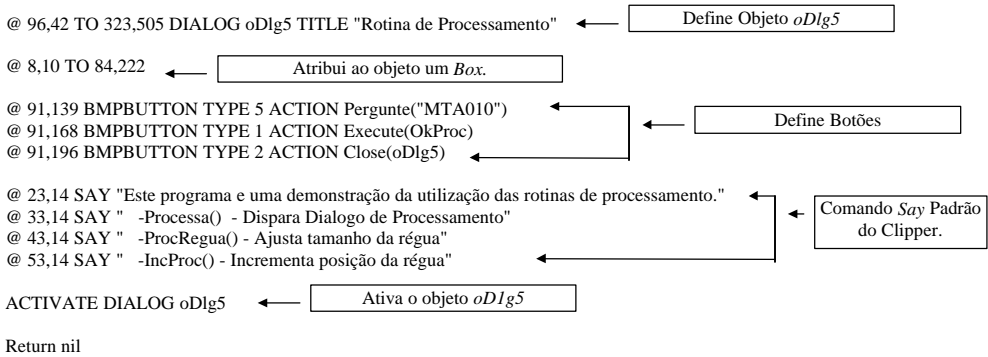
Return

/*

Função	ProcDemo
Autor	Ary Medeiros
Data	15/02/96
Descrição	Exemplo de utilização de rotinas de processamento

*/

Function ProcDemo

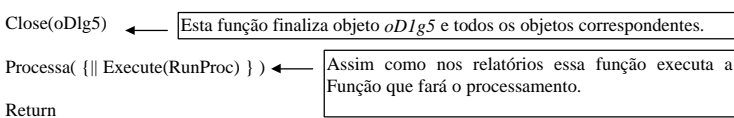


/*

Função	OkProc
Autor	Ary Medeiros
Data	15/02/96
Descrição	Confirma o processamento

*/

Function OkProc




```
/*
```

Função	OkProc
Autor	Ary Medeiros
Data	15/02/96
Descrição	Executa o processamento

```
*/
Function RunProc
ProcRegua(1000) ← Inicializa a régua padrão de processamentos pode ser
                    utilizada no RdMake Dos/Windows.

For i:= 1 to 1000
  IncProc() ← Contador para demonstrar a função que incrementa a
Next          régua padrão previamente definida.

Return
```

```
/*
```

Função	DlgDinam
Autor	Ary Medeiros
Data	15/02/96
Descrição	Exemplo de rotina de refresh

```
*/
Function DlgDinam

cSay1 := "Demo 01" ← Define variáveis.
cSay2 := "Demo 02"
cGet1 := "Get 01"
cGet2 := "Get 02"

@ 96,42 TO 323,505 DIALOG oDlg6 TITLE "Rotina de Refresh" ← Define Objeto oDlg6

@ 8,10 TO 84,222 ← Atribui um Box ao objeto.

@ 91,139 BUTTON "Refresh" Size 70,20 ACTION Execute(RefrDlg) ← Define um botão.

@ 23,14 SAY cSay1 ← Comando Say Padrão do Clipper.
@ 33,14 SAY cSay2

@ 43,14 Get cGet1 ← Comando Get, não é necessário utilizar o Comando Read, no Windows o Get
@ 53,14 Get cGet2 ← estará ativo quando o objeto ao qual foi atribuído for Ativado.

ACTIVATE DIALOG oDlg6 CENTERED ← Ativa o objeto oDlg6

Return
```

Function RefrDlg

```
cSay1 := IIF(cSay1=="Demo 01","Demo 03","Demo 01")  
cSay2 := IIF(cSay2=="Demo 02","Demo 04","Demo 02")  
cGet1 := IIF(cGet1=="Get 01","Get 03","Get 01")  
cGet2 := IIF(cGet2=="Get 02","Get 04","Get 02")
```

Redefine variáveis.

dlgRefresh(oDlg6) ←

Return

Esta função é utilizada para atualizar as informações do formulário ativo, em Dos sempre que for necessário atualizar um valor na tela é necessário utilizar o Comando Say, em Windows basta apenas "atualizar" (Refresh), pois, o objeto "Say" já está ativo no formulário.

```

/*
-----
--| Programa      | RDDEM02 | Autor  | Luiz Carlos Vieira | Data | Wed 23/09/98 |--
--| Descrição    | Demo. Interpretador XBase para Windows. Versao 2 com mais |
--|              | exemplos, de comandos e uso de metodos e propriedades. |
--| Uso          | Especifico para clientes Microsiga |
-----

*/

//-----
//| Variaveis do programa |
//-----

oRDDEM02 := NIL
oTimer    := NIL

cMsg      := Space(20)
cMsg      := cMsg + "Esta versao do RDDEM0 demonstra novos conceitos e o uso de novas "
cMsg      := cMsg + "possibilidades para os comandos do interpretador para Windows. "
cMsg      := cMsg + "Além disso, demonstra formas de acesso a propriedades de objetos "
cMsg      := cMsg + "e métodos, o que facilita a programação nesse ambiente. Escolha "
cMsg      := cMsg + "através "
cMsg      := cMsg + "das duas listas abaixo as demonstrações que deseja visualizar e "
cMsg      := cMsg + "pressione "
cMsg      := cMsg + "o botão EXECUTAR. "
cMsg      := oemToAnsi(cMsg)

nPosMsg   := 1

bTimer    := { || cTopBar := Substr(cMsg, nPosMsg, 65) , ;
                  nPosMsg := If(nPosMsg > Len(cMsg), 1, nPosMsg + 1), ;
                  ObjectMethod(oGt, "Refresh()") }

cTopBar   := Space(65)

nSource   := 0
aSource   := {"Uso de Senhas", "Novos Dialogos", "Refresh de Objetos", "Campos MEMO"}
nTarget   := 0
aTarget   := {}

//-----
//| Criacao do dialogo principal |
//-----

@ 105,074 To 304,716 Dialog oRDDEM02 Title "RDDEM02 - Demonstração do Interpretador XBase
for Windows"

//-----
//| Define o timer que ira executar por detras do dialogo |
//-----

oTimer := IW_Timer(100, bTimer)
ObjectMethod(oTimer, "Activate()")

//-----
//| Objetos do dialogo principal |
//-----

@ 020,003 Say oemToAnsi("Demonstrações disponiveis:")
@ 020,133 Say oemToAnsi("Executar Demonstrações:")

```

```

//-----
//| Nova clausula disponivel para todos os comandos -> OBJECT <NOME> |
//-----

@ 004,000 Get cTopBar      When .F.                Object oGt
@ 030,004 ListBox nSource Items aSource Size 86,65 Object oSource
@ 029,133 ListBox nTarget Items aTarget Size 85,65 Object oTarget

@ 030,093 Button OemToAnsi("_Adicionar >>") Size 36,16 Action Execute(AddDemo) Object
oBtnAdd
@ 048,093 Button OemToAnsi("<< _Remover") Size 36,16 Action Execute(RemoveDemo) Object
oBtnRem

@ 030,250 Button OemToAnsi("_Executar") Size 36,16 Action Execute(RunDemos)

@ 082,277 BmpButton Type 1 Action Close(oRDDEMO2)

Activate Dialog oRDDEMO2

//-----
//| Funcao para acesso aos metodos de um objeto -> OBJECTMETHOD |
//-----

//-----
//| Libera o timer |
//-----

ObjectMethod(oTimer,"DeActivate()")
ObjectMethod(oTimer,"Release()")

Return

/* /

-----
--| Função      | ADDDEMO | Autor | Luiz Carlos Vieira | Data | Wed 23/09/98 |--
--| Descrição   | Adiciona o item da lista Source para a lista Target |
--| Uso         | Específico para clientes Microsiga |
-----

/* /

Function AddDemo
  If nSource != 0
    aAdd(aTarget,aSource[nSource])
    ObjectMethod(oTarget,"SetItems(aTarget)")
    nNewTam := Len(aSource) - 1
    aSource := aSize(aDel(aSource,nSource),nNewTam)
    ObjectMethod(oSource,"SetItems(aSource)")
  Endif
Return

```

```

/* /
-----
-- Função      REMOVEDEMO  Autor   Luiz Carlos Vieira  Data   Wed 23/09/98 --
-- Descrição    Remove um item da lista Target e adiciona na lista Source --
-- Uso          Específico para clientes Microsiga --
-----
/* /

```

```

function RemoveDemo
  If nTarget != 0
    aAdd(aSource, aTarget[nTarget])
    ObjectMethod(oSource, "SetItems(aSource)")
    nNewTam := Len(aTarget) - 1
    aTarget := aSize(aDel(aTarget, nTarget), nNewTam)
    ObjectMethod(oTarget, "SetItems(aTarget)")
  Endif
Return

```

```

/* /
-----
-- Função      RUNDEMOS    Autor   Luiz Carlos Vieira  Data   Wed 23/09/98 --
-- Descrição    Executa os demos da lista Target --
-- Uso          Específico para clientes Microsiga --
-----
/* /

```

```

Function RunDemos

If Len(aTarget) != 0
  For nDemo := 1 To Len(aTarget)

    cDemo := AllTrim(Upper(aTarget[nDemo]))

    Do Case
      Case cDemo == "USO DE SENHAS"
        Execute(FunSenhas)
      Case cDemo == "NOVOS DIALOGOS"
        Execute(FunNovos)
      Case cDemo == "REFRESH DE OBJETOS"
        Execute(FunRefresh)
      Case cDemo == "CAMPOS MEMO"
        Execute(FunMEMO)
    EndCase

  Next nDemo
Endif

Return

```

```

/*/
-----
-- Função      FUNSENHAS  Autor      Luiz Carlos Vieira  Data      Wed 23/09/98
-- Descrição    Demonstracao do uso de senhas
-- Uso          Especifico para clientes Microsiga
-----
/*/

Function FunSenhas

cSenha      := Space(200)
cConteudo   := Space(200)

@ 115,085 To 267,727 Dialog oSenhas Title "Demonstraç,õ de Objetos GET com senhas"
@ 002,002 To 038,315
@ 009,008 Say 0emfoAnsi(' Para se trabalhar com caixas de ediç,õ que n,õ exibem o contefdo
digitado, basta acrescentar a cl usula')
@ 020,008 Say 0emfoAnsi(' PASSWORD ao comando GET do programa. Por exemplo: @00,00 Get
cSenha Picture "@!" Valid .T. PASSWORD')
@ 042,004 Say 0emfoAnsi("Senha      ")
@ 057,004 Say 0emfoAnsi("Conteúdo: ")
@ 042,042 Get cSenha      Picture "@S40" Valid .T.  PASSWORD      Object oSenha
@ 057,042 Get cConteudo Picture "@S40" When .F.      Object oConteudo

@ 059,277 BmpButton Type 1 Action Close(oSenhas)

oAtuCont := Iw_Timer(100,{|| cConteudo := cSenha , ObjectMethod(oConteudo,"Refresh()") })
ObjectMethod(oAtuCont,"Activate()")

Activate Dialog oSenhas

ObjectMethod(oAtuCont,"DeActivate()")
ObjectMethod(oAtuCont,"Release()")

Return

/*/
-----
-- Função      FUNNOVOS  Autor      Luiz Carlos Vieira  Data      Wed 23/09/98
-- Descrição    Demonstracao de diãlogos diversi ficados.
-- Uso          Especifico para clientes Microsiga
-----
/*/

Function FunNovos

nTipo := 1
aTipo := {"Selecao de Arquivo","Selecao de Cores"}

@ 150,133 To 361,716 Dialog oDialogos Title "Demonstraç,õ de Diãlogos Diversificados"
@ 003,002 To 039,282

@ 011,008 Say 0emfoAnsi("Al,m dos diãlogos para simples exibiç,õ de mensagens, existem os
diãlogos para seleç,õ de arquivos,")
@ 023,008 Say 0emfoAnsi("entradas de dados, etc, que facilitam a vida do programador
RDMAKE.")
@ 049,003 Say 0emfoAnsi("Tipo do Diãlogo: ")

```



```

@ 116,090 To 416,707 Dialog oRefresh Title OemToAnsi("Refresh de Objetos")
@ 003,002 To 040,305

@ 012,008 Say OemToAnsi("Alguns dos objetos no Windows, como o SAY e o GET, possuem um
m, todo REFRESH(). Desse modo")
@ 024,009 Say OemToAnsi("pode-se atualizar as informações desses objetos sem forçar a
atualização de todo o diálogo.")

@ 055,005 Say OemToAnsi("Hora atual:")

oTimerHora := Iw_Timer(500, { || ObjectMethod(oHoraAtual, "SetText(Time())") })
ObjectMethod(oTimerHora, "Activate()")

@ 055,040 Say OemToAnsi(Time())      Object oHoraAtual
@ 080,005 Say OemToAnsi("1o. Get: ") Object oLabel1
@ 104,005 Say OemToAnsi("2o. Get: ") Object oLabel2
@ 080,040 Get cPrimeiro              Object oGet1
@ 104,040 Get cSegundo              Object oGet2

@ 088,262 Button OemToAnsi("_Trocar") Size 36,16 Action Execute(FRTroca)
@ 132,263 BmpButton Type 1 Action Close(oRefresh)

```

Activate Dialog oRefresh

```

ObjectMethod(oTimerHora, "DeActivate()")
ObjectMethod(oTimerHora, "Release()")

```

Return

```

/*/

```

Função	FRTROCA	Autor	Luiz Carlos Vieira	Data	Wed 23/09/98
Descrição	Faz a troca dos objetos no dialogo da funcao FUNREFRESH				
Uso	Específico para clientes Microsiga				

```

/*/

```

Function FRTroca

```

cAux1      := cPrimeiro
cAux2      := cSegundo

```

```

cSegundo := cAux1
cPrimeiro := cAux2

```

```

ObjectMethod(oGet1, "Refresh()")
ObjectMethod(oGet2, "Refresh()")

```

```

If lPrimeiro
    ObjectMethod(oLabel1, "SetText(' 2o. Get: ')")
    ObjectMethod(oLabel2, "SetText(' 1o. Get: ')")
    lPrimeiro := .F.
Else
    ObjectMethod(oLabel1, "SetText(' 1o. Get: ')")
    ObjectMethod(oLabel2, "SetText(' 2o. Get: ')")
    lPrimeiro := .T.

```

```

Endif
Return

```



```

/*/
-----
--|Função      |FUNMEMD|Autor|Luiz Carlos Vieira|Data|Thu 24/09/98|--
--|Descrição   |Demo. da edicao de campos memo|--
--|Uso        |Específico para clientes Microsiga|--
-----

/*/

Function FunMEMD

cTexto := ""
cF0pen := ""

@ 116,090 To 416,707 Dialog oDlgMemo Title "Demonstraç,ão de Campos MEMD - Editor de
Arquivos Texto"
@ 003,002 To 040,305

@ 012,008 Say 0emToAnsi("Para editar um campo MEMD, ou mesmo o conteúdo de um arquivo
texto, utiliza-se o próprio")
@ 024,009 Say 0emToAnsi("comando GET, adicionando as cláusulas SIZE e MEMD. Por exemplo:
@10,10 GET cVar Size 50,50 MEMD")

@ 045,005 Say 0emToAnsi("Arquivo: <SEM NOME>" + Space(100)) Object oNome
@ 055,005 Get cTexto Size 250,080 MEMD Object oMemo

@ 045,263 Button 0emToAnsi("_Abrir...") Size 36,16 Action Execute(FRABre)
@ 063,263 Button 0emToAnsi("_Fechar") Size 36,16 Action Execute(FRFechar)
@ 081,263 Button 0emToAnsi("_Salvar") Size 36,16 Action Execute(FRSalva)
@ 099,263 Button 0emToAnsi("_Salvar Como...") Size 36,16 Action Execute(FRSalvaComo)

@ 132,263 BmpButton Type 1 Action Close(oDlgMemo)

Activate Dialog oDlgMemo

Return

/*/
-----
--|Função      |FRABRE|Autor|Luiz Carlos Vieira|Data|Thu 24/09/98|--
--|Descrição   |Rotina para a abertura do arquivo texto na FunMEMD|--
--|Uso        |Específico para clientes Microsiga|--
-----

/*/

Function FRABre

cF0pen := cGetFile("Arquivos Texto|*.TXT|Todos os Arquivos|*. *", 0emToAnsi("Abrir
Arquivo..."))
If !Empty(cF0pen)
    cTexto := MemoRead(cF0pen)
    ObjectMethod(oMemo, "Refresh()")
    ObjectMethod(oNome, "SetText('Arquivo: ' + cF0pen)")
EndIf

Return

```

```

/* /
-----
-- Função      FRFECHA      Autor      Luiz Carlos Vieira      Data      Thu 24/09/98 --
-- Descrição   Rotina para fechamento do arquivo texto em FunMEMO
-- Uso         Especifico para clientes Microsiga
-----
/* /

Function FRFecha
  cTexto := ""
  cF0pen := ""
  ObjectMethod(oMemo, "Refresh() ")
  ObjectMethod(oNome, "SetText(' Arquivo: <SEM NOME>') ")
Return

/* /
-----
-- Função      FRSalva      Autor      Luiz Carlos Vieira      Data      Thu 24/09/98 --
-- Descrição   Rotina para salvar o arquivo texto em FunMEMO
-- Uso         Especifico para clientes Microsiga
-----
/* /

Function FRSalva
If !Empty(cF0pen)
  MemoWrit(cF0pen, cTexto)
Endif
Return

/* /
-----
-- Função      FRSALVACOM      Autor      Luiz Carlos Vieira      Data      Thu 24/09/98 --
-- Descrição   Rotina para salvar arquivo texto com outro nome em FunMEMO
-- Uso         Especifico para clientes Microsiga
-----
/* /

Function FRSalvaComo
cAux      := cF0pen
cF0pen := cGetFile("Arquivos Texto|*.TXT|Todos os Arquivos|*. *", 0emToAnsi("Salvar Arquivo
Como..."))
If !Empty(cF0pen)
  MemoWrit(cF0pen, cTexto)
  ObjectMethod(oNome, "SetText(' Arquivo: ' +cF0pen) ")
Else
  cF0pen := cAux
Endif
Return

```

Referências e Abreviaturas

SGBD	<i>Sistema Gerenciador de Banco de Dados.</i> É o aplicativo que gerencia os dados e controla o acesso aos mesmos.
SQL	<i>Structured Query Language</i> – Linguagem de Consulta Estruturada. É uma linguagem de consulta ágil e estruturada, bastante difundida entre gerenciadores de banco de dados relacionais.
Oracle, MSSQL Server ou DB2	Estes são sistemas de gerenciamento de banco de dados relacional em bastante evidência no mercado atual. Oracle é marca registrada da Oracle Inc., MSSQL da Microsoft e DB2 da IBM.
ADS	<i>Advantage DataBase Server.</i> É marca registrada da Advantage, e é um <i>Driver</i> para acesso à base de dados padrão <i>xBase</i> em ambiente de rede.