Protheus com Paulo Bindo

Esta foto representa os profissionais de TI no dia a dia

Baixe todos os materiais antes de começar a instalação, e seja um bombeiro eficiente, leia as documentações sugeridas

BOAS PRÁTICAS PROGRAMAÇÃO

Palavras Maiúsculas

A regra é utilizar caracteres em maiúsculo para:

Constantes:

```
#Define NUMLINES 60
#Define NUMPAGES 1000
```

Variáveis de memória:

```
M->CT2_CRCONV
M->CT2_MCONVER := CriaVar("CT2_CONVER")
```

Campos:

```
SC6->C6_NUMPED
```

Querys:

```
SELECT * FROM...
```

Notação Húngara

A notação húngara consiste em adicionar os prefixos aos nomes de variáveis, de modo a facilmente se identificar seu tipo. Isto facilita na criação de códigos-fonte extensos, pois usando a Notação Húngara, você não precisa ficar o tempo todo voltando à definição de uma variável para se lembrar qual é o tipo de dados que deve ser colocado nela. Variáveis devem ter um prefixo de Notação Húngara em minúsculas, seguido de um nome que identifique a função da variável, sendo que a inicial de cada palavra deve ser maiúscula.

É obrigatória a utilização desta notação para nomear variáveis.

Protheus com Paulo Bindo

Notação	Tipo de dado	Exemplo
a	Array	aValores
b	Bloco de código	bSeek
c	Caracter	cNome
d	Data	dDataBase
l	Lógico	lContinua
n	Numérico	nValor
o	Objeto	oMainWindow
x	Indefinido	xConteudo

Palavras Reservadas

AADD	DTOS	INKEY	REPLICATE	VAL
ABS	ELSE	INT	RLOCK	VALTYPE
ASC	ELSEIF	LASTREC	ROUND	WHILE
AT	EMPTY	LEN	ROW	WORD
BOF	ENDCASE	LOCK	RTRIM	YEAR
BREAK	ENDDO	LOG	SECONDS	CDOW
ENDIF	LOWER	SELECT	CHR	EOF
LTRIM	SETPOS	CMONTH	EXP	MAX
SPACE	COL	FCOUNT	MIN	SQRT

CTOD	FIELDNAME	MONTH	STR	DATE
FILE	PCOL	SUBSTR	DAY	FLOCK
PCOUNT	TIME	DELETED	FOUND	PROCEDURE
TRANSFORM	DEVPOS	FUNCTION	PROW	TRIM
DOW	IF	RECCOUNT	TYPE	DTOC
IIF	RECNO	UPPER	TRY	AS
CATCH	THROW			

Palavras reservadas não podem ser utilizadas para variáveis, procedimentos ou funções;
Funções reservadas são pertencentes ao compilador e não podem ser redefinidas por uma aplicação;
Todos os identificadores, que começarem com dois ou mais caracteres "_", são utilizados como identificadores internos e são reservados.
Identificadores de escopo PRIVATE ou PUBLIC utilizados em aplicações específicas, desenvolvidas por ou para clientes, devem ter sua identificação iniciada por um caractere "_".

Protheus com Paulo Bindo

HISTÓRIA DO ADVPL

A Linguagem ADVPL teve seu início em 1994, sendo na verdade uma evolução na utilização de linguagens no padrão xBase pela Microsiga Software S.A. (Clipper, Visual Objects e depois FiveWin). Com a criação da tecnologia Protheus, era necessário criar uma linguagem que suportasse o padrão xBase para a manutenção de todo o código existente do sistema de ERP Siga Advanced. Foi então criada a linguagem chamada Advanced Protheus Language. O ADVPL é uma extensão do padrão xBase de comandos e funções, operadores, estruturas de controle de fluxo e palavras reservadas, contando também com funções e comandos disponibilizados pela Microsiga que a torna uma linguagem completa para a criação de aplicações ERP prontas para a Internet. Também é uma linguagem orientada a objetos e eventos, permitindo ao programador desenvolver aplicações visuais e criar suas próprias classes de objetos. Quando compilados, todos os arquivos de código tornam-se unidades de inteligência básicas, chamados APO´s (de Advanced Protheus Objects). Tais APO´s são mantidos em um repositório e carregados dinamicamente pelo PROTHEUS Server para a execução. Como não existe a linkedição, ou união física do código compilado a um determinado módulo ou aplicação, funções criadas em ADVPL podem ser executadas em qualquer ponto do Ambiente Advanced Protheus.
O compilador e o interpretador da linguagem ADVPL é o próprio servidor PROTHEUS (PROTHEUS Server), e existe um Ambiente visual para desenvolvimento integrado (PROTHEUSIDE), em que o código pode ser criado, compilado e depurado.

Os programas em ADVPL podem conter comandos ou funções de interface com o usuário. De acordo com tal característica, tais programas são subdivididos nas seguintes categorias

Protheus com Paulo Bindo

DECLARAÇÃO E ATRIBUIÇÃO DE VARIÁVEIS

Tipo de Dados

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

Numérico

O ADVPL não diferencia valores inteiros de valores com ponto flutuante, portanto podem-se criar variáveis numéricas com qualquer valor dentro do intervalo permitido. Os seguintes elementos são do tipo de dado numérico:
2
43.53
0.5
0.00001
1000000
Uma variável do tipo de dado numérico pode conter um número de dezoito dígitos, incluindo o ponto flutuante, no intervalo de 2.2250738585072014 E–308 até 1.7976931348623158 E+308.

Lógico

Valores lógicos em ADVPL são identificados através de .T. ou .Y. para verdadeiro e .F. ou .N. para falso (independentemente se os caracteres estiverem em maiúsculo ou minúsculo).

4.4. Caractere

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

"Olá !!!"

'Esta é uma string'
"Esta é 'outra' string"
Uma variável do tipo caractere pode conter strings com no máximo 1 MB, ou seja, 1048576 caracteres.

Data

O ADVPL tem um tipo de dados específico para datas. Internamente as variáveis deste tipo de dado são armazenadas como um número correspondente à data Juliana.
Variáveis do tipo de dados Data não podem ser declaradas diretamente, e sim através da utilização de funções específicas como, por exemplo, CTOD() que converte uma string para data.

Array

O Array é um tipo de dado especial. É a disposição de outros elementos em colunas e linhas. O ADVPL suporta arrays unidimensionais (vetores) ou multidimensionais (matrizes). Os elementos de um array são acessados através de índices numéricos iniciados em 1, identificando a linha e coluna para quantas dimensões existirem.
Arrays devem ser utilizadas com cautela, pois se forem muito grandes podem exaurir a memória do servidor.

Bloco de Código

O bloco de código é um tipo de dado especial. É utilizado para armazenar instruções escritas em ADVPL que poderão ser executadas posteriormente.

Declaração de variáveis

Variáveis de memória são um dos recursos mais importantes de uma linguagem. São áreas de memória criadas para armazenar informações utilizadas por um programa para a execução de tarefas. Por exemplo, quando o usuário digita uma informação qualquer, como o nome de um produto, em uma tela de um programa esta informação é armazenada em uma variável de memória para posteriormente ser gravada ou impressa.
A partir do momento que uma variável é criada, não é necessário mais se referenciar ao seu conteúdo, e sim ao seu nome. O nome de uma variável é um identificador único o qual deve respeitar um máximo de 10 caracteres. O ADVPL não impede a criação de uma variável de memória cujo nome contenha mais de 10 caracteres, porém apenas os 10 primeiros serão considerados para a localização do conteúdo armazenado.
Portanto se forem criadas duas variáveis cujos 10 primeiros caracteres forem iguais, como nTotalGeralAnual e nTotalGeralMensal, as referências a qualquer uma delas no programa resultarão o mesmo, ou seja, serão a mesma variável:

```
nTotalGeralMensal := 100
nTotalGeralAnual  := 300
Alert("Valor mensal: " + cValToChar(nTotalGeralMensal))
```

Quando o conteúdo da variável nTotalGeralMensal é exibido, o seu valor será de 300. Isso acontece porque no momento que esse valor foi atribuído à variável nTotalGeralAnual, o ADVPL considerou apenas os 10 primeiros caracteres (assim como o faz quando deve exibir o valor da variável nTotalGeralMensal), ou seja, considerou-as como a mesma variável. Assim o valor original de 100 foi substituído pelo de 300.

Protheus com Paulo Bindo

OPERADORES LINGUAGEM ADVPL

Operadores Matemáticos

Os operadores utilizados em ADVPL para cálculos matemáticos são:

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
**	Exponenciação
ou	
^	
%	Módulo (Resto da Divisão)

Operadores de String

Os operadores utilizados em ADVPL para tratamento de caracteres são:

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

Operadores Relacionais

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

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

Protheus com Paulo Bindo

Operadores Lógicos

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

.And.	E lógico
.Or.	OU lógico
.Not. ou !	NÃO lógico

Operadores de Atribuição

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

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

Operadores Especiais

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

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

Protheus com Paulo Bindo

CONVERSÕES ENTRE TIPOS DE VARIÁVEIS

Conversões entre tipos de variáveis

As funções mais utilizadas nas operações entre conversão entre tipos de variáveis são:

CTOD()

Sintaxe	CTOD(cData)
Descrição	Realiza a conversão de uma informação do tipo caractere no formato "DD/MM/AAAA", para uma variável do tipo data.
Exemplo	CtoD("01/12/16") + 5 -> 06/12/2016

DTOC()

Sintaxe	DTOC(dData)
Descrição	Realiza a conversão de uma informação do tipo data para em caractere, sendo o resultado no formato "DD/MM/AAAA".

Exemplo	DtoC(Date()) -> data da maquina em formato "DD/MM/AA" caracter.

DTOS()

Sintaxe	DTOS(dData)
Descrição	Realiza a conversão de uma informação do tipo data em um caractere, sendo o resultado no formato "AAAAMMDD".
Exemplo	Converte uma data , no formato string "AAAAMMDD" DtoS (Date()) -> "20161212"

STOD()

Sintaxe	STOD(sData)
Descrição	Realiza a conversão de uma informação do tipo caractere "AAAAMMDD", com conteúdo no formato em data
Exemplo	Converte data em string, no formato Stod (Date()) -> ""AAAAMMDD"

CVALTOCHAR()

Sintaxe	CVALTOCHAR(nValor)
Descrição	Realiza a conversão de uma informação do tipo numérico em uma string, sem a adição de espaços a informação.
Exemplo	cValTochar(50) -> "50"

STR()

Sintaxe	STR(nNumero, nTamanho, nDecimal)
Descrição	Realiza a conversão de uma informação do tipo numérico em uma string, adicionando espaços à direita.
Exemplo	Str(700, 8, 2) -> " 700.00"

STRZERO()

Sintaxe	STRZERO(nNumero, nTamanho, nDecimal)
Descrição	Realiza a conversão de uma informação do tipo numérico em uma string, adicionando zeros à esquerda do número convertido, de forma que a string gerada tenha o tamanho especificado no parâmetro.
Exemplo	StrZero(60, 10) -> "0000000060"

VAL()

Sintaxe	VAL(cValor)
Descrição	Realiza a conversão de uma informação do tipo caractere em numérica.
Exemplo	Val("1234") -> 1234

FUNÇÕES DE MANIPULAÇÃO DE ARRAYS

Funções de manipulação de arrays

A linguagem ADVPL possui diversas funções que auxiliam na manipulação de arrays, dentre as quais podemos citar as mais utilizadas:

ARRAY()

Sintaxe	ARRAY(nLinhas, nColunas)
Descrição	A função Array() é utilizada na definição de variáveis de tipo array, como uma opção a sintaxe, utilizando chaves ("{}").
Exemplo	aVetor := Array(3) -> aVetor[1] aVetor[2] aVetor[3] aMatriz := Array(3,2) -> aMatriz[1,1] aMatriz[1,2] aMatriz[2,1] aMatriz[2,2] aMatriz[3,1] aMatriz[3,2]

AADD()

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

Exemplo	Local aVetor := {}
	AADD(aVetor, "Maria") // Adiciona um elemento no array
	AADD(aVetor, "Jose") // Adiciona um elemento no array
	AADD(aVetor, "Marcio") // Adiciona um elemento no array
	
	Local aMatriz := {}
	AADD(aMatriz, { "Maria" ,29,"F"}) // Adiciona um elemento no array
	AADD(aMatriz, { "Jose" ,42,"M"}) // Adiciona um elemento no array
	AADD(aMatriz, { "Marcio" ,53"M"}) // Adiciona um elemento no array

Protheus com Paulo Bindo

ACLONE()

Sintaxe	AADD(aArray)
Descrição	A função ACLONE() realiza a cópia dos elementos de um array para outro array integralmente.
Exemplo	altens := ACLONE(aDados)

ADEL()

Sintaxe	ADEL(aArray, nPosição)
Descrição	A função ADEL() permite a exclusão de um elemento do array. Ao efetuar a exclusão de um elemento, todos os demais são reorganizados de forma que a ultima posição do array passará a ser nula, sendo necessário informar um novo valor pela função aSize() ou Ains()
Exemplo	Local aVetor := {"1","2","3","4","5","6","7","8","9"} ADEL(aVetor,1) // Será removido o primeiro elemento do array. ->> aVetor{"2","3","4","5","6","7","8","9",NIL}

ASIZE()

Sintaxe	ASIZE(aArray, nTamanho)
Descrição	A função ASIZE permite a redefinição da estrutura de um array pré-existente, adicionando ou removendo itens do mesmo.
Exemplo	Local aVetor := {"1","2","3","4","5","6","7","8","9"} aSize(aVetor,8) // Será restruturado o array com 8 elementos. ->> aVetor{"1","2","3","4","5","6","7","8" }

AINS()

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

Exemplo	Local aVetor := {"1","2","3","4","5","6","7","8","9"} aIns(aVetor,1) // Será restruturado o array com 8 elementos. -> aVetor{Nil,"1","2","3","4","5","6","7","8" }

ASORT()

Sintaxe	ASORT(aArray, nInicio, nItens, bOrdem)
Descrição	A função ASORT() permite que os itens de um array sejam ordenados, a partir de um critério pré-estabelecido.
Exemplo	LOCAL aVetor := { "A", "D", "C", "B" } ASORT(aVetor) -> { "A", "B", "C", "D" } ASORT(aVetor,,, { |x, y| x > y }) -> { "D", "C", "B", "A" } Local aMatriz := { { "Fipe", 9.3 }, { "IPC", 8.7 }, { "DIEESE", 12.3 } } ASORT(aMatriz, , , { | x,y | x[2] > y[2] }) // Com base na ordenação acima, a ordem fica: DIEESE 12.3 Fipe 9.3 IPC 8.7

ASCAN()

Sintaxe	ASCAN(aArray, bSeek)
Descrição	A função ASCAN() permite que seja identificada a posição do array que contém uma determinada informação, através da análise de uma expressão descrita em um bloco de código.
Exemplo	LOCAL aVetor := { "Java", "AdvPL", "C++" } nPos := ASCAN(aVetor, "AdvPL" } -> 2 posição no array Local aMatriz := { { "Fipe", 9.3 }, { "IPC", 8.7 }, { "DIEESE", 12.3 } } nPos := ASCAN(aVetor,{ | x | x[1] == "AdvPL" } -> 3 posição no array

Protheus com Paulo Bindo

BLOCOS DE CÓDIGO

Blocos de Código

Diferentemente de uma matriz, não se pode acessar elementos de um bloco de código, através de um índice numérico. Porém blocos de código são semelhantes a uma lista de expressões, e a uma pequena função.
Ou seja, podem ser executados. Para a execução, ou avaliação de um bloco de código, deve-se utilizar a função Eval():
nRes := Eval(B) ==> 20
Essa função recebe como parâmetro um bloco de código e avalia todas as expressões contidas neste bloco de código, retornando o resultado da última expressão avaliada.

Funções para manipulação de blocos de código

EVAL()

Sintaxe	EVAL(bBloco, xParam1, xParam2, xParamZ)
Descrição	A função EVAL() é utilizada para avaliação direta de um bloco de código, utilizando as informações disponíveis no momento de sua execução. Esta função permite a definição e passagem de diversos parâmetros que serão considerados na interpretação do bloco de código.
Exemplo:	nInt := 10 bBloco := {|N| x:= 10, y:= x*N, z:= y/(x*N)} nValor := EVAL(bBloco, nInt) O retorno será dado pela avaliação da ultima ação da lista de expressões, no caso "z". Cada uma das variáveis definidas, em uma das ações da lista de expressões, fica disponível para a próxima ação. Desta forma temos: N → recebe nInt como parâmetro (10) X → tem atribuído o valor 10 (10) Y → resultado da multiplicação de X por N (100) Z → resultado da divisão de Y pela multiplicação de X por N (100 / 100) → 1

DBEVAL()

Sintaxe	Array(bBloco, bFor, bWhile)
Descrição	A função DBEval() permite que todos os registros, de uma determinada tabela, sejam analisados e para cada registro será executado o bloco de código definido.
Exemplo	dbSelectArea("SX5") dbSetOrder(1) dbGotop() While !Eof() .And. X5_FILIAL == xFilial("SX5") .And.; X5_TABELA <=cTab nCnt++ dbSkip() EndDo O mesmo pode ser re-escrito com o uso da função DBEVAL(): dbEval({|x| nCnt++ },,{||X5_FILIAL==xFilial("SX5") .And. X5_TABELA<= cTab })

AEVAL()

Sintaxe	AEVAL(aArray, bBloco, nInicio, nFim)
Descrição	A função AEVAL() permite que todos os elementos de um determinada array sejam analisados e para cada elemento será executado o bloco de código definido.
Exemplo	Considerando o trecho de código abaixo: AADD(aCampos,"A1_FILIAL") AADD(aCampos,"A1_COD") SX3->(dbSetOrder(2)) For nX:=1 To Len(aCampos) SX3->(dbSeek(aCampos[nX])) aAdd(aTitulos,AllTrim(SX3->X3_TITULO)) Next nX O mesmo pode ser re-escrito com o uso da função AEVAL(): aEval(aCampos,{|x| SX3->(dbSeek(x)),IIF(Found(), AAdd(aTitulos,AllTrim(SX3->X3_TITULO)))})

FUNÇÕES DE ACESSO E MANIPULAÇÃO DE DADOS

Funções de acesso e manipulação de dados

As funções de acesso e manipulação de dados, descritas neste tópico, são as classificadas anteriormente como funções genéricas da linguagem ADVPL, permitindo que as mesmas sejam utilizadas independentemente da base de dados para a qual a aplicação ERP está configurada.

DBRLOCK()

Sintaxe	DBRLOCK(xIdentificador)
Descrição	Função de base de dados, que efetua o lock (travamento) do registro identificado pelo parâmetro xIdentificador. Este parâmetro pode ser o Recno() para tabelas em formado ISAM, ou a chave primária para bancos de dados relacionais. Se o parâmetro xIdentificador não for especificado, todos os locks da área de trabalho serão liberados, e o registro posicionado será travado e adicionado em uma lista de registros bloqueados.

DBCLOSEAREA()

Sintaxe	DbCloseArea()
Descrição	Permite que um alias presente na conexão seja fechado, o que viabiliza novamente seu uso em outro operação. Este comando tem efeito apenas no alias ativo na conexão, sendo necessária sua utilização em conjunto com o comando DbSelectArea().

DBCOMMIT()

Sintaxe	DBCOMMIT()
Descrição	Efetua todas as atualizações pendentes na área de trabalho ativa.

DBCOMMITALL()

Sintaxe	DBCOMMITALL()
Descrição	Efetua todas as atualizações pendentes em todas as área de trabalho, em uso pela thread (conexão) ativa.

DBDELETE()

Sintaxe	DbDelete()
Descrição	Efetua a exclusão lógica do registro posicionado na área de trabalho ativa, sendo necessária sua utilização em conjunto com as funções RecLock() e MsUnLock().

DBGOTO()

Sintaxe	DbGoto(nRecno)
Descrição	Move o cursor da área de trabalho ativa para o record number (recno) especificado, realizando um posicionamento direto, sem a necessidade de uma busca (seek) prévia.

DBGOTOP()

Sintaxe	DbGoTop()
Descrição	Move o cursor da área de trabalho ativa para o primeiro registro lógico.

DBGOBOTTON()

Sintaxe	DbGoBotton()
Descrição	Move o cursor da área de trabalho ativa para o último registro lógico.

DBRLOCKLIST()

Sintaxe	DBRLOCKLIST()
Descrição	Retorna um array contendo o record number (recno) de todos os registros, travados da área de trabalho ativa.

DBSEEK() E MSSEEK()

Sintaxe	DbSeek(cChave, lSoftSeek, lLast)
Descrição	DbSeek: Permite posicionar o cursor da área de trabalho ativo no registro com as informações especificadas na chave de busca, fornecendo um retorno lógico indicando se o posicionamento foi efetuado com sucesso, ou seja, se a informação especificada na chave de busca foi localizada na área de trabalho. MsSeek(): Função desenvolvida pela área de Tecnologia da Microsiga, a qual possui as mesmas funcionalidades básicas da função DbSeek(), com a vantagem de não necessitar acessar novamente a base de dados para localizar uma informação já utilizada pela thread (conexão) ativa.

DBSKIP()

Sintaxe	DbSkip(nRegistros)
Descrição	Move o cursor do registro posicionado para o próximo (ou anterior, dependendo do parâmetro), em função da ordem ativa para a área de trabalho.

DBSELECTAREA()

Sintaxe	DbSelectArea(nArea | cArea)
Descrição	Define a área de trabalho especificada como sendo a área ativa. Todas as operações subseqüentes que fizerem referência a uma área de trabalho para utilização, a menos que a área desejada seja informada explicitamente.

DBSETFILTER()

Sintaxe	DbSetFilter(bCondicao, cCondicao)

Descrição	Define um filtro para a área de trabalho ativa, o qual pode ser descrito na forma de um bloco de código ou através de uma expressão simples.

DBSETORDER()

Sintaxe	DbSetOrder(nOrdem)
Descrição	Define qual índice será utilizado pela área de trabalho ativa, ou seja, pela área previamente selecionada através do comando DbSelectArea(). As ordens disponíveis no Ambiente Protheus são aquelas definidas no SINDEX /SIX, ou as ordens disponibilizadas por meio de índices temporários.

DBORDERNICKNAME()

Sintaxe	DbOrderNickName(NickName)
Descrição	Define qual índice criado pelo usuário será utilizado. O usuário pode incluir os seus próprios índices e no momento da inclusão deve criar o NICKNAME para o mesmo.

DBUNLOCK()

Sintaxe	DBUNLOCK()
Descrição	Mesma funcionalidade da função UNLOCK(), só que recomendada para ambientes de rede nos quais os arquivos são compartilhados. Libera o travamento do registro posicionado na área de trabalho ativa e confirma as atualizações efetuadas naquele registro.

DBUNLOCKALL()

Sintaxe	DBUNLOCKALL()
Descrição	Libera o travamento de todos os registros de todas as áreas de trabalho disponíveis na thread (conexão) ativa.

DBUSEAREA()

Sintaxe	DbUseArea(lNovo, cDriver, cArquivo, cAlias, lComparilhado,; lSoLeitura)
Descrição	Define um arquivo de base de dados como uma área de trabalho disponível na aplicação.

MSUNLOCK()

Sintaxe	MsUnLock()
Descrição	Libera o travamento (lock) do registro posicionado, confirmando as atualizações efetuadas neste registro.

RECLOCK()

Sintaxe	RecLock(cAlias,lInclui)
Descrição	Efetua o travamento do registro posicionado na área de trabalho ativa, permitindo a inclusão ou

	alteração das informações do mesmo.

RLOCK()

Sintaxe	RLOCK() → lSucesso
Descrição	Efetua o travamento do registro posicionado na área de trabalho ativa.

SELECT()

Sintaxe	Select(cArea)
Descrição	Determina o número de referência de um determinado alias em um ambiente de trabalho. Caso o alias especificado não esteja em uso no Ambiente, será retornado o valor 0 (zero).

SOFTLOCK()

Sintaxe	SoftLock(cAlias)
Descrição	Permite a reserva do registro posicionado na área de trabalho ativa de forma que outras operações, com exceção da atual, não possam atualizar este registro. Difere da função RecLock() pois não gera uma obrigação de atualização, e pode ser sucedido por ele. Na aplicação ERP Protheus, o SoftLock() é utilizado nos browses, antes da confirmação da operação de alteração e exclusão, pois neste momento a mesma ainda não foi efetivada, mas outras conexões não podem acessar aquele registro pois o mesmo está em manutenção, o que implementa da integridade da informação.

UNLOCK()

Sintaxe	UNLOCK()
Descrição	Libera o travamento do registro posicionado na área de trabalho ativa e confirma as atualizações efetuadas naquele registro.

CONTROLE DE SEMÁFORO E NUMERAÇÃO SEQUENCIAL

Sintaxe	GETSXENUM(cAlias, cCampo, cAliasSXE, nOrdem)
Descrição	Obtém o número seqüência do alias especificado no parâmetro, através da referência aos arquivos de sistema SXE/SXF ou ao servidor de numeração, quando esta configuração está habilitada no Ambiente Protheus.

CONFIRMSX8()

Sintaxe	**CONFIRMSX8()**
Descrição	Confirma o número alocado através do último comando GETSXENUM().

ROLLBACKSXE()

Sintaxe	ROLLBACKSXE()
Descrição	Descarta o número fornecido pelo último comando GETSXENUM(), retornando a numeração disponível para outras conexões.

lockByname()

https://tdn.totvs.com/pages/releaseview.action?pageId=6814894

UnlockByname()

https://tdn.totvs.com/pages/releaseview.action?pageId=6814897

CUSTOMIZAÇÃO DE PARÂMETROS

GETMV()

Sintaxe	GETMV(cParametro)
Descrição	Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Caso o parâmetro não exista, será exibido um help do sistema informando a ocorrência.

GETNEWPAR()

Sintaxe	GETNEWPAR(cParametro, cPadrao, cFilial)
Descrição	Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Caso o parâmetro não exista, será exibido um help do Sistema informando a ocorrência. Difere do SuperGetMV() pois considera que o parâmetro pode não existir na versão atual do Sistema, e por conseqüência não será exibida a mensagem de help.

PUTMV()

Sintaxe	PUTMV(cParametro, cConteudo)
Descrição	Atualiza o conteúdo do parâmetro especificado no arquivo SX6, de acordo com as parametrizações informadas.

SUPERGETMV()

Sintaxe	SUPERGETMV(cParametro , lHelp , cPadrao , cFilial)
Descrição	Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Caso o parâmetro não exista, será exibido um help do Sistema informando a ocorrência. Difere do GetMv() pois os parâmetros consultados são adicionados em uma área de memória, que permite que em uma nova consulta não seja necessário acessar e pesquisar o parâmetro na base de dados.

CUSTOMIZAÇÃO DE CAMPOS

EXISTCHAV()

Sintaxe	ExistChav(cAlias, cConteudo, nIndice)
Descrição	Retorna .T. ou .F. se o conteúdo especificado existe no alias especificado. Caso exista será exibido um help de Sistema com um aviso informando da ocorrência. Função utilizada normalmente para verificar se um determinado código de cadastro já existe na tabela na qual a informação será inserida, como por exemplo, o CNPJ no cadastro de clientes ou fornecedores.

EXISTCPO()

Sintaxe	ExistCpo(cAlias, cConteudo, nIndice)
Descrição	Retorna .T. ou .F. se o conteúdo especificado não existe no alias especificado. Caso não exista será exibido um help de Sistema com um aviso informando da ocorrência. Função utilizada normalmente para verificar se a informação digitada em um campo, a qual depende de outra tabela, realmente existe nesta outra tabela, como por exemplo, o código de um cliente em um pedido de venda.

NAOVAZIO()

Sintaxe	NaoVazio()
Descrição	Retorna .T. ou .F. se o conteúdo do campo posicionado no momento não está vazio.

NEGATIVO()

Sintaxe	Negativo()
Descrição	Retorna .T. ou .F. se o conteúdo digitado para o campo é negativo.

PERTENCE()

Sintaxe	Pertence(cString)
Descrição	Retorna .T. ou .F. se o conteúdo digitado para o campo está contido na string, definida como parâmetro da função. Normalmente utilizada em campos com a opção de combo, pois caso contrário seria utilizada a função ExistCpo().

POSITIVO()

Sintaxe	Positivo()
Descrição	Retorna .T. ou .F. se o conteúdo digitado para o campo é positivo.

TEXTO()

Sintaxe	Texto()
Descrição	Retorna .T. ou .F. se o conteúdo digitado para o campo contém apenas números ou alfanuméricos.

VAZIO()

Sintaxe	Vazio()
Descrição	Retorna .T. ou .F. se o conteúdo do campo posicionado no momento está vazio.

INTERFACE VISUAL

BUTTON()

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

MSDIALOG()

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

MSGET()

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

SAY()

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

SBUTTON()

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

Tipos de botões:

https://tdn.totvs.com/display/tec/SButton?showChildren=false

https://tdn.totvs.com/display/tec/TButton

REGUAS DE PROCESSAMENTO

15.1.1. RptStatus()

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

Sintaxe: RptStatus(bAcao, cMensagem)

SETREGUA()

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

Sintaxe: SetRegua(nMaxProc)

Parâmetros:

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

INCREGUA()

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

Parâmetros:

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

15.1.2. Processa()

Régua de processamento simples, com apenas um indicador de progresso, utilizada no processamento de rotinas.
Sintaxe: Processa(bAcao, cMensagem)
Retorno: Nil

SETPROC()

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

Sintaxe: Processa(nMaxProc)

Parâmetros:

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

INCPROC()

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

Sintaxe: IncProc(cMensagem)

Parâmetros:

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

15.1.3. MsNewProcess().

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

Sintaxe: MsNewProcess():New(bAcao, cMensagem)
Retorno: oProcess → objeto do tipo MsNewProcess()

Parâmetros:

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

Métodos:

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

15.1.4. MsAguarde().

Indicador de processamento sem incremento

Sintaxe: Processa(bAcao, cMensagem, cTitulo)
Retorno: Nil

Parâmetros:

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

15.1.5. MsgRun().

Indicador de processamento sem incremento.

Sintaxe: Processa(cMensagem, cTitulo, bAcao)
Retorno: Nil

FWMSGRUN - https://tdn.totvs.com/display/public/PROT/FWMsgRun

Trabalhe e se organize como um bombeiro, você será um profissional muito requisitado!

Protheus com Paulo Bindo