

Sumário

	Conteúdo
Capítulo 01 – ADVPL.....	6
Capítulo 02 – Programação.....	7
Capítulo 03 – Estruturação	9
Capítulo 04 – Documentação	11
Capítulo 05 – Funcionalidades	13
Criação de Variáveis	13
Capítulo 06 – Operadores Básicos.....	15
Matemáticos	15
String	15
Relacionais	15
Lógicos	15
Atribuições	16
Especiais	16
Procedências	17
Macro Substituição	18
Exercícios	18
Capítulo 07 - Utilização de Vetores	20
Cuidados com Vetores	21
Matrizes como Estruturas	21
Exercícios	21
Capítulo 08 – Controle de Fluxo	22
Comandos de Repetição	22
Comandos de Decisão	24
Problemas Comuns	26
Exercícios	27
Capítulo 09 – Development Studio.....	29
Configuração de Ambiente	29
Criação de Projetos	30
Utilizando assistentes de Programas	31
Compilação	34
Depuração	35
Capítulo 10 – Customização.....	37
Parâmetros	37
Tabelas	39

Perguntas.....	41
Expressões ADVPL.....	41
Pontos de Entrada	41
Dicionário de Dados Ativos	43
Gatilhos	43
Capítulo 11 – Conceitos de Filiais	44
Arquivos Compartilhados	44
Arquivos Exclusivos.....	44
Capítulo 12 – Programando	46
Criando um Programa Simples	46
Incluindo Item no Menu	50
Exercícios	52
Capítulo 13 – Funções Pré-Existentes.....	54
Manipulação de Stings.....	54
Manipulação de Dados.....	57
Manipulação de Números	59
Manipulação de Vetores	60
Capítulo 14 – Tratamento de Base de Dados.....	62
Criando Arquivos.....	62
Criando Indices Temporários.....	62
Posicionamento de Registros	63
Funções de Base de Dados	65
Capítulo 15 – Blocos de Códigos.....	68
Definição	68
Funções de Bloco.....	69
Capítulo 16 – Funções Diversas.....	71
Funções de Ambientes.....	71
Funções de Servidor	71
Funções de Comunicação Server x Client.....	73
Funções de Servidor	74
Capítulo 17 – Tela de Padrão Microsiga	76
AxCadastro()	76
Pergunte()	77
PutSx1().....	78
mBrowse()	79
Modelo 2.....	83
Modelo 3.....	84

Exercício.....	87
Capítulo 18 – MSEXCAUTO.....	87
Exemplo de Automação de Rotinas.....	90
Exercícios	95

Capítulo 01 – ADVPL

ADVPL (Advanced Protheus Language) surgiu com a tecnologia Protheus em 1994, derivada do Clipper e bibliotecas Fivewin, incorpora o padrão xBase para a manutenção de todo o código já existente do sistema de ERP Siga Advanced, possui comandos e funções, operadores, estruturas de controle de fluxo e palavras reservadas, contando também com funções e comandos criados pela própria Microsiga que a torna uma linguagem completa para a criação de aplicações ERP. Também é uma linguagem orientada a objetos e eventos, permitindo ao programador desenvolver aplicações visuais e criar suas próprias classes de objetos.

A Microsiga criou seu próprio ambiente de desenvolvimento (IDE) que edita programas, compila, interpreta e depura as funções de usuário que ficam armazenadas nas unidades de inteligência básicas, chamados APO's (de *Advanced Protheus Objects*), que ficam guardados em um repositório carregado pelo Protheus no momento de sua execução. Toda função criada em AdvPL pode ser executada em qualquer ponto do ambiente Protheus.

Os programas em AdvPL são subdivididos nas seguintes categorias:

- **Interface com o Usuário**

São funções que realizem algum tipo de interface remota utilizando o protocolo de comunicação do Protheus. Podem ser criadas rotinas para a customização desde processos adicionais até mesmo relatórios. A grande vantagem é aproveitar todo o ambiente montado pelos módulos, utilizando telas e funções padronizadas. Com o ADVPL é possível criar uma aplicação do começo ao fim.

- **Processos**

São funções sem interface com o usuário, são rotinas como processos internos ou Jobs. Essas rotinas são executadas de forma transparente ao usuário.

Existem rotinas que são responsáveis por iniciar os processos executados através dos meios disponíveis de integração e conectividade no Protheus, subdivididas em:

- o **Programação de RPC**

Executada através de uma biblioteca de funções disponíveis diretamente pelo Server do Protheus ou através de aplicações externas escritas em outras linguagens. Essa execução pode ser feita em outros servidores Protheus através de conexão TCP.

- o **Programação Web**

São requisições http feitas em ADVPL que são executadas como um servidor Web, como processos individuais, enviando ou recebendo resultado das funções. Lembrando que as funções não devem ter comandos de interface com o usuário, nesse caso utiliza-se arquivos HTML contendo código ADVPL, conhecidos como ADVPL ASP, para a criação de páginas dinâmicas.

- o **Programação TelNet**

O Protheus Server pode emular um terminal TelNet, através da execução de rotinas escritas em ADVPL, cuja interface final será um terminal TelNet ou um coletor de dados móvel.

Capítulo 02 – Programação

Um programa de computador nada mais é do que um algoritmo escrito numa linguagem de computador com o objetivo de executar determinada tarefa. Esses comandos são gravados em um arquivo texto que é transformado em uma linguagem executável por um computador através da compilação. A compilação substitui os comandos que estão numa linguagem que nós entendemos por linguagem de máquina. No caso do AdvPL, esse código compilado será executado pelo Protheus Server a partir do Repositório.

Para programar em qualquer linguagem devem ser seguidas regras da linguagem, a qual chamamos de sintaxe da linguagem, se não obedecida essa sintaxe o programa causará erros que podem ser de compilação ou de execução.

Compilação não permite nem que o código seja compilado, são comandos especificados incorretamente, operadores utilizados de forma errada, sintaxe incorreta, dentre outros.

Execução são os que ocorrem no momento da execução. Podem ocorrer por muitas razões, dentre elas, funções não existentes, ou variáveis não criadas ou não inicializadas, etc.

Linhas de Programa

As linhas existentes dentro de um arquivo texto de código de programa podem ser:

Linhas de comando possuem os comandos ou instruções que serão executadas.

Linhas de comentário possuem um texto qualquer, que não são executadas é muito utilizado para documentação e facilitar o entendimento do programa. Podemos comentar informações no programa utilizando os seguintes caracteres:

&&	&&Programa para recalculo do custo médio
//	// Programa para recalculo do custo médio

Podemos documentar blocos de texto inicializando com /* e finalizando com */

```
/*  
Programa para recalculo do custo médio  
Autor: Biale ADVPL e Consultoria em Informática  
Data: 15 de dezembro de 2006  
*/
```

Linhas Mistas possuem comandos ou instruções juntamente com comentários

Local nSoma := 0 //Utilizada para totalizar o relatório.

Tamanho de Linha

Linha física, delimitada pela quantidade de caracteres que pode ser digitado no editor de textos utilizado. Cada linha física é separada por um <Enter>.

Linha lógica, é aquela considerada para a compilação como uma única linha de comando. Quando queremos que mais de uma linha física componha a mesma linha de comando utilizamos o ; (ponto e virgula)

```
If !Empty(cNome) .And. !Empty(cEnd) .And. ; <enter>  
    !Empty(cTel) .And. !Empty(cFax) .And. ; <enter>  
    !Empty(cEmail)  
    GravaDados(cNome,cEnd,cTel,cFax,cEmail)  
Endif
```

Capítulo 03 – Estruturação

A linguagem ADVPL não tem padrões rígidos de estrutura do programa, mas normalmente utilizamos a seguinte estruturação:

Identificação: Área reservada para documentação do programa, como finalidade, data, autor, etc.

```
/*  
|Funcao para separação de números pares e impares  
|Autor: Cristiane C. Figueiredo  
|Data : 01/03/06  
|*/
```

Inicialização: Área reservada a declarações de variáveis, abertura de arquivos. Apesar de em ADVPL ser permitido o uso de variáveis não declaradas anteriormente, é importante declara-las aqui para tornar o programa mais legível.

```
User Function fSeparaNum  
Local nBI  
Local cBIImpares := "" //Numeros Impares  
Local cBIPares := "" //Números Pares
```

Corpo do programa: Área reservada para a lógica do programa.

```
For nBI:=1 to 12  
  If mod(nBI,2)= 0  
    cBIPares += Alltrim(str(nBI)) + " "  
  Else  
    cBIImpares += Alltrim(str(nBI)) + " "  
  Endif  
Next
```

Encerramento: Área reservada ao fechamento dos arquivos abertos, mostrar o resultado do processamento e encerramento do programa.

```
// Exibe em tela o resultado encontrado  
Msginfo("Pares " + cBIPares + " Impares " + cBIImpares)  
  
// Termina o programa  
Return
```

Geralmente utilizamos na declaração de variável um prefixo de acordo com a tipo de cada

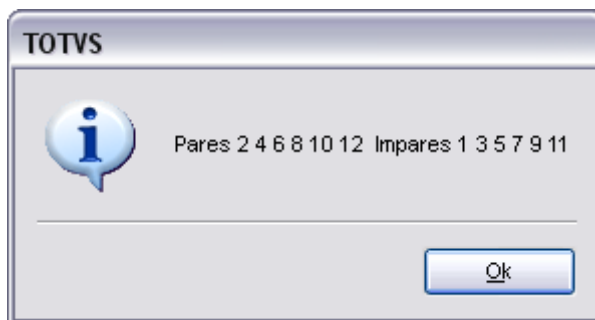
variável e um nome sugestivo, em seguida atribuímos valores às variáveis. A atribuição também pode ser feita no momento da declaração.

Veja como ficou nosso exemplo obedecendo as características da estrutura do código escrito em ADVPL:

```

/*****
|Funcao para separação de números pares e impares
|Autor: Cristiane C. Figueiredo
|Data : 01/03/10|
\*****/
*****/
User Function fSeparaNum
Local nBI
Local cBIImpares := "" //Numeros Impares
Local cBIPares := "" //Números Pares
For nBI:=1 to 12
    If mod(nBI,2)== 0
        cBIPares += Alltrim(str(nBI)) + " "
    Else
        cBIImpares += Alltrim(str(nBI)) + " "
    Endif
Next
//Exibe em tela o resultado encontrado
Msginfo("Pares " + cBIPares + " Impares " + cBIImpares)
Return

```



Capítulo 04 – Documentação

Toda customização deve ser bem documentada, com informações relevantes e conceituais. O cabeçalho das funções deve conter o nome do autor, data de criação, uma descrição sumária de sua funcionalidade, a sintaxe e a descrição dos argumentos de entrada e saída. Pode também ser colocada a localização no menu, agendamento, etc.

```

/*=====
=====|
|Função   |ADV0070A |Autor|Cristiane C. Figueiredo   | Data |23/07/04|
|=====|=====|
|=====|
|Descrição|Verifica se campo do E2 pode ser alterado ou não.      |
|=====|=====|
|=====|
|Modulo   | SIGAFIN                                     |
|=====|=====|
|=====|
|OBS      | Especifico para BIALE                                     |
|=====|=====|
|=====|
|Alterações solicitadas                                     |
|=====|=====|
|=====|
|Descrição                                     |Sol.por|Atend.por |Data   |
|=====|=====|
|=====|
|Incluida verificação de chamada por MsExecAuto                                     |
|ISInCallStack('msexecauto').               |Gerente|Lucas Borges|30/01/07|
|=====|=====|
=====*/

```

- **Documentação em Funções:**

User Function ADV0070A(nCampo)

Comandos...

Return

- **Documentação de solicitação de Customizações:**

Também é necessária para uma melhor visualização a documentação e de todas as solicitações efetuadas pelos usuários ou gerentes com as devidas assinaturas.

Veja na página seguinte o exemplo de uma documentação de solicitação de customização.

Dados da Requisição

Data Solicitação	<data da solicitação>
Área	<nome da área solicitante/centro de custo>
Responsável	<nome do solicitante responsável>
Analista de Negócio	< nome do analista de negócio >
Volume de esforço Previsto	<quantidade de horas necessárias para o desenvolvimento/testes>
Data de entrega Prevista	<data de entrega prevista (uso da área de TI)>

Objetivo

Esta rotina tem por objetivo ... (descrever aqui o objetivo macro que a rotina deverá atender. Neste espaço não devem ser descritas as regras de negócio, pois há uma seção mais adiante pra atender a esta necessidade. Descrever o objetivo em uma linguagem que o usuário entenda).

Programas envolvidos (análise inicial)**Comentários/Observações do Responsável pela Customização****Aprovações**

Nome	Assinatura	Data

Capítulo 05 – Funcionalidades

Em ADVPL utilizamos funções da seguinte forma:

Function	Limitada a programas padrões, não permitindo ser compilada por clientes.
User Function	Funções de usuário, podendo ser utilizadas para fazer customizações.
Static Function	Funções chamadas pelas User Functions e pelas Functions.

Especificamente Functions e User Functions podem ser chamadas de qualquer ponto do Sistema, se estiverem compiladas no repositório.

Criação de Variáveis

As variáveis devem ser declaradas no início da função que for utilizá-la.

As variáveis declaradas em um programa ou função, são visíveis de acordo com o escopo onde são definidas. Como também do escopo depende o tempo de existência das variáveis.

A definição do escopo de uma variável é efetuada no momento de sua declaração.

- **Identificadores**

Public: Visível a partir do momento que é criada até o termino da aplicação, não deve ser utilizada para qualquer coisa pois ocupa espaço de memória.

Private: Visível na função que a criou e nas demais chamadas por ela, até que a função que a criou seja finalizada, e são criadas automaticamente quando não declaradas.

Local: Visíveis somente na função que a criou, variáveis locais ocupam pouco espaço de memória. E são criadas automaticamente quando provierem de parametrização de função.

Tipos

O tipo de variável identifica sua utilização na função. Toda variável deve estar tipada durante sua criação.

É convencional utilizarmos letras iniciais em variáveis para que possamos saber se o seu tipo é numérico, caracter, data ou lógico.

As variáveis têm os seguintes tipos:

Tipo	Exemplo	Descritivo
Caracter	cVar	Aceita números e letras
Númérica	nVar	Aceita apenas números
Data	dVar	Aceita apenas Datas
Lógica	LVar	Aceita Verdadeira ou Falsa
Array	aVar	Aceita Vetores
Objeto	oVar	Usada em objetos

Variáveis e nomes de campos

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

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

cRes := MEMVAR->NOME

cRes := FIELD->NOME

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

cRes := CLIENTES->NOME

• Inicialização de Variáveis

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

• Utilização da Função CRIAVAR()

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

Sintaxe: uRet := CriaVar(cCampo, lIniPad)

URet	tipo de retorno de acordo com o dicionário de dados, considerando inicializador
CCampo	Nome do campo
LiniPad	Indica se considera (.T.) ou não (.F.) o inicializador padrao (X3_RELACAO)

Capítulo 06 – Operadores Básicos

Matemáticos

Os operadores utilizados para cálculos matemáticos são:

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

String

Os operadores utilizados 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)

Relacionais

Os operadores utilizados para operações e avaliações relacionais são:

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

Lógicos

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

.And.	E
.Or.	OU
.Not. ou !	NÃO

Atribuições

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

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

Especiais

Além dos operadores comuns, o AdvPl possui alguns operadores especiais. Estas são suas finalidades:

()	Utilizados para agrupar elementos em uma expressão priorizando essas condições, ao qual damos o nome de " ordem de precedência " da avaliação da expressão (da mesma forma que as regras matemáticas). Também servem para envolver os argumentos de uma função
[]	Elemento de Matriz. Utilizados para especificar um elemento específico de uma matriz. Por exemplo, aArray[1,5], refere-se ao elemento da matriz aArray na linha 1, coluna 5.
{ }	Definição de Matriz, Constante ou Bloco de Código. Por exemplo: aArray := {1,2,3,4,5} cria uma matriz chamada aArray com cinco elementos.
->	Identificador de Apelido (Alias). identifica um campo de um arquivo diferenciando-o de uma variável. Exemplo: SA1->A1_NOME, o campo A1_NOME da tabela SA1, mesmo que haja uma variável com o nome A1_NOME, a partir desse Alias o campo da tabela que será acessado.
&	Identifica uma avaliação de expressão através macrossubstituição.
@	Passagem de parâmetro por referência. Utilizado para indicar que durante a passagem de uma variável para uma função ou procedimento ela seja tomada como uma referência e não como valor.
	Passagem de parâmetro por valor. Utilizado para indicar que durante a passagem de uma variável para uma função ou procedimento ela seja tomada como um e valor não como referência.

Procedências

A ordem de precedência, ou nível de prioridade de execução, dos operadores é a seguinte:

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

Caso as expressões complexas sejam de mesmo nível a avaliação será feita da esquerda para direita, exceto as matemáticas que obedecem o seguinte:

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

Exemplo:

Local nVar := 15+12/3+7*3-2^3

Calcula-se a exponenciação: $2^3 = 8$

Calcula-se divisão: $12/3 = 4$

Calcula-se a multiplicação: $7*3 = 21$

Efetua a soma e subtração: $15 + 4 + 21 - 8$

O resultado desta expressão é 32

Para alterarmos a precedência utilizamos os parênteses, o grupo mais a esquerda será avaliado primeiro e assim por diante, como também o mais interno para o mais externo, caso haja parênteses dentro de parênteses.

Local nVar := (15+12)/(3+7)*3-2^3

Calcula-se a exponenciação: $2^3 = 8$

Calcula-se o primeiro grupo de parênteses: $15+12 = 27$

Calcula-se o segundo grupo: $3+7=10$

Calcula-se a divisão: $27/10 = 2.7$

Calcula-se a multiplicação: $2.7 * 3 = 8.1$

Efetua a soma e subtração: $8.1-8$

O resultado desta expressão é 0.1

Macro Substituição

O operador de macro substituição, simbolizado pelo "&" comercial (&), é utilizado para a avaliação de expressões em tempo de execução, ou seja, o resultado será sobre o conteúdo do resultado da variável.

```
cConteudo := "MACRO"
```

```
cVar := "cConteudo"
```

```
cVar1 := &cVar
```

cVar1 será igual a "MACRO"

é o mesmo que:

```
cVar1 := cConteudo
```

```
cVar := "MACRO"
```

O variável cVar é substituída pelo seu conteúdo (cConteudo) e é essa variável que será resolvida pelo AdvPL.

Outro Exemplo:

```
nVar := 20
```

```
cVar := "nVar + 35"
```

```
nResult := &cVar
```

```
nResult := nVar + 35
```

```
nResult := 20 + 35
```

```
nResult := 55
```

```
IVar := .T.
```

```
cVar := "IVar .or. .F."
```

```
IResult := &cVar
```

```
IResult := IVar .or. ..F.
```

```
IResult := .T. .or. .F.
```

```
IResult := .T.
```

Exercícios

1. De acordo com a instrução fazendo um teste de mesa, informe ao final qual o tipo e conteúdo de cada variável criada.

```
01 User Function fProcVar()
```

```
02 Local cVal, nTotGer, IVar1, cTem, ICont, ITudo
```

```
03 Local cNom := ""
```

```
04 Local nVal := 5
```

```
05 Private nVal1:= 7 + 6, IVal, IQuas, nValor
```

```
06 Public IVar := nVal == 5
```

```
07 nVal1 += 15
```



```
08  nValor := (((nVal1- nVal)/2) * 3^2 + 1 ) * -1
09  nTempo := "nValor * 3"
10  nTotGer:= &nTempo
11  nTotGer++
12  cTem  := "São Paulo"
13  cTem  += ", 15 de Dezembro"
14  lCont  := "Dez" $ cTem
15  lTudo  := cTem = "São"
16  lQuas  := cTem == "São Paulo"
17  lVal   := nVal > 5
18  lValc  := (nVal1 - 8) <= 13
19
20  fMudaFun(@nValor, lTudo, lQuas, lVal)
21  msginfo(nValor)
22 Return
23
24 Static Function fMudaFun(_val, _lTudo, _lQuas, _lVal)
25  msginfo(nVal1)
26 Return
27
28 User Function fContinua()
29  Msginfo(lVar)
30 Return
```

2. De acordo com o exercício anterior responda Verdadeiro ou Falso, nas afirmações a seguir e justifique:

a. () _lTudo na linha 25 será nula.

b. () nTempo é do tipo caracter.

c. () lCont tem o conteúdo Verdadeiro.

d. () lVar não está visível na linha 29

e. () lTudo na linha 25 tem seu identificador Local

f. () cNom na linha 18 é nula

g. () lVar não pode ser acessada na linha 25

h. () nValor pode ser acessada na linha 25

Capítulo 07 - Utilização de Vetores

Um vetor é uma lista com linhas e colunas. Comparando com uma tabela, as linhas são os registros e as colunas são os campos, podemos então dizer que é uma tabela virtual. Vetores podem ser chamados de Array ou matrizes.

Cada item do vetor é referenciado pela sua posição na lista, iniciando pelo número 1, veja exemplo:

AArray	Col1	Col2	Col3
Lin1	A	B	C
Lin2	D	E	F
Lin3	G	H	I

A montagem deste array em AdvPL se dá da seguinte forma:

```
Local aArray //Declaração da variável
aArray := {"A", "B", "C"}, {"D", "E", "F"}, {"G", "H", "I"} //Atribuição da matriz à
variável
```

Para exibirmos seu conteúdo utilizamos a seguinte sintaxe:

```
MsgInfo(aArray[nLin][nCol])
```

Onde:

nLin -> Linha onde se encontra o conteúdo que se quer exibir

nCol -> Coluna onde se encontra o conteúdo que se quer exibir

```
MsgInfo(aArray[2][3]) //Exibe o Conteúdo da 2ª. Linha e 3ª. Coluna, (F)
```

```
MsgInfo(aArray[1][1]) //Exibe o Conteúdo da 1ª. Linha e 1ª. Coluna, (A)
```

```
MsgInfo(aArray[3][2]) //Exibe o Conteúdo da 3ª. Linha e 2ª. Coluna, (H)
```

Para atribuírmos outro conteúdo utilizamos a seguinte sintaxe:

```
aArray[nLin][nCol] := cVar
```

Onde:

nLin -> Linha onde se encontra o conteúdo que se quer alterar

nCol -> Coluna onde se encontra o conteúdo que se quer alterar

cVar -> Novo Conteúdo

```
aArray[2][3] := "X" // Altera o Conteúdo da 2ª. Linha e 3ª. Coluna de F para X
```

```
aArray[1][1] := "Y" // Altera o Conteúdo da 1ª. Linha e 1ª. Coluna de A para Y
```

```
aArray[3][2] := "Z" // Altera o Conteúdo da 3ª. Linha e 2ª. Coluna de H para Z
```

Após essa alteração a nossa lista virtual ficará assim:

AArray	Col1	Col2	Col3
Lin1	Y	B	C
Lin2	D	E	X
Lin3	G	Z	I

Cuidados com Vetores

Vetores são listas de elementos virtuais, utilizamos então a memória é necessária para armazenar estas informações. Como as matrizes podem ser multidimensionais, a memória necessária será a multiplicação do número de itens em cada dimensão da matriz, portanto precisamos usá-las com certo conscientemente.

Quando seu conteúdo por algum motivo começar a ficar muito complexo, é recomendável utilizarmos outros recursos, com criação de tabelas temporárias.

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

Matrizes como Estruturas

Uma característica interessante do AdvPL é que uma matriz pode conter qualquer coisa: números, datas, lógicos, caracteres, objetos, etc. E ao mesmo tempo. Em outras palavras, os elementos de uma matriz não precisam ser necessariamente do mesmo tipo de dado.

Exercícios

1. Criar um vetor de 5 linhas com 3 Colunas e dar o nome de aArray1.
2. Criar um vetor de 10 linhas com 1 Coluna e dar o nome de aArray2.
3. Atribuir ao Vetor a Array1 linha 1 coluna 1 o conteúdo "X"
4. Atribuir ao Vetor a Array1 linha 3 coluna 2 o conteúdo 9
5. Atribuir ao Vetor a Array1 linha 5 coluna 3 o conteúdo .T.
6. Atribuir ao Vetor a Array1 linha 1 coluna 3 o conteúdo do aArray2 Linha 5 coluna 1.

Capítulo 08 – Controle de Fluxo

Podemos mudar a seqüência de fluxo de execução de um programa baseado em condições lógicas e a repetição da execução de pedaços de código quantas vezes forem necessárias para tal utilizamos estruturas de controle com um identificador de início e um de fim, e o que deve ser executado deve estar entre estes identificadores. Essas estruturas em AdvPL estão divididas em :

Estruturas de Repetição:

Execução de uma seção de código mais de uma vez. Ex. For / While

Estruturas de Decisão:

Execução de uma seção de código de acordo com uma condição lógica. Ex. If / Case.

Comandos de Repetição

Em AdvPL existem dois comandos para a repetição de seções de código. O comando FOR...NEXT e o comando WHILE...ENDDO.

FOR...NEXT

A estrutura de controle FOR...NEXT repete uma seção de código em um número determinado de vezes.

Sintaxe

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

Comandos...

[EXIT]

[LOOP]

NEXT

Parâmetros

Variável	Variável utilizada como contador. Se a variável não existir, será criada como uma variável privada.
<i>nValorInicial</i> TO <i>nValorFinal</i>	<i>nValorInicial</i> é o valor inicial para o contador; <i>nValorFinal</i> é o valor final para o contador. Pode-se utilizar valores numéricos literais, variáveis ou expressões, contanto que o resultado seja do tipo de dado numérico.
STEP <i>nIncremento</i>	<i>nIncremento</i> é a quantidade que será incrementada ou decrementada no contador após cada execução da seção de comandos. Se o valor de <i>nIncremento</i> for negativo, o contador será decrementado. Se a cláusula STEP for omitida, o contador será incrementado em 1. Pode-se utilizar valores numéricos literais, variáveis ou expressões, contanto que o resultado seja do tipo de dado numérico.

Comandos	Especifica uma ou mais instruções de comando AdvPl que serão executadas.
EXIT	Finaliza a repetição da seção de comandos imediatamente.
LOOP	Retorna ao Início do FOR sem executar o restante dos comandos, incrementando ou decrementando normalmente.

Este exemplo separa os números pares e ímpares em variáveis para imprimir ou mostrar em tela.

```

For nBI:=1 to 12
  If mod(nBI,2)== 0
    cBIPares += Alltrim(str(nBI)) + " "
  Else
    cBIImpares += Alltrim(str(nBI)) + " "
  Endif
Next

```

WHILE...ENDDO

A estrutura de controle WHILE...ENDDO, repete uma seção de código enquanto uma determinada condição resultar em verdadeiro (.T.)

Sintaxe

WHILE *IExpressao*

Comandos...

[EXIT]

[LOOP]

ENDDO

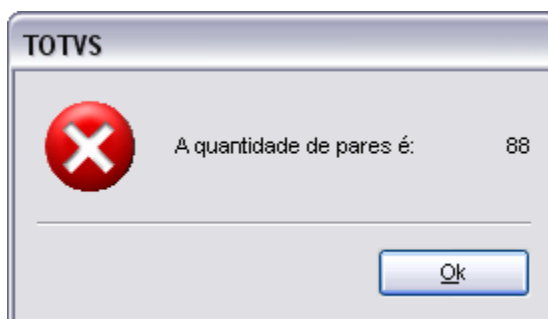
Parâmetros

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

	executados.
--	-------------

Este exemplo separa os números pares e ímpares em variáveis para imprimir ou mostrar em tela.

```
Local nSomaPar := 0, nNumber := 350
  nNumber := Int(nNumber / 2)
  While nNumber > 0
    nSomaPar++
    nNumber:= nNumber - 2
  Enddo
  Alert( "A quantidade de pares é: " + str(nSomaPar) )
Return
```



Comandos de Decisão

Em AdvPL existem dois comandos de decisão. O comando IF...ENDIF e o comando CASE...ENDCASE.

IF...ENDIF

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

Sintaxe

```
IF IExpressao
  Comandos
[ELSE
  Comandos...]
ENDIF
```

Parâmetros

IExpressao	Especifica uma expressão lógica que é avaliada. Se IExpressao resultar em verdadeiro (.T.), qualquer comando seguinte ao IF e antecedente ao ELSE ou ENDIF (o que ocorrer primeiro) será executado.
------------	---

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

Exemplo

```
Local dVencto := CTOD("31/12/2010")
  If Date() > dVencto
    Alert("Titulo vencido!")
  Else
    Alert("Titulo a vencer!")
  Endif
Return
```

DO CASE...ENDCASE

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

Sintaxe

```
DO CASE
  CASE IExpressao1
    Commandos
  [CASE IExpressao2
    Commandos
  ...
  CASE IExpressaoN
    Commandos]
  [OTHERWISE
    Commandos]
ENDCASE
```

Parâmetros

<i>CASE</i> <i>IExpressao1</i> <i>Comandos...</i>	Quando a primeira expressão CASE resultante em verdadeiro (.T.) for encontrada, o conjunto de comandos seguinte é executado. A execução do conjunto de comandos continua até que a próxima cláusula CASE, OTHERWISE ou ENDCASE seja encontrada. Ao terminar de executar esse
---	--

	conjunto de comandos, a execução continua com o primeiro comando seguinte ao ENDCASE. Se uma expressão CASE resultar em falso (.F.), o conjunto de comandos seguinte a esta até a próxima cláusula é ignorado. Após a execução, qualquer outra expressão CASE posterior é ignorada (mesmo que sua avaliação resultasse em verdadeiro).
OTHERWISE <i>Comandos</i>	Se todas as expressões CASE forem avaliadas como falso (.F.), a cláusula OTHERWISE determina se um conjunto adicional de comandos deve ser executado. Se essa cláusula for incluída, os comandos seguintes serão executados e então o programa continuará com o primeiro comando seguinte ao ENDCASE. Se a cláusula OTHERWISE for omitida, a execução continuará normalmente após a cláusula ENDCASE.

Exemplo

Local nMes := Month(Date())

Local cPeriodo := ""

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

Return

Problemas Comuns

- Looping, é uma repetição infinita, ou seja, por um erro na lógica do programa, o mesmo não finaliza, com isso o processo pode utilizar todo o recurso do servidor e diminuir a performance ou até mesmo travá-lo, tornando necessário o reinício do sistema.

Exemplo 1:

```
dbSeek(xFilial("SE1")+DTOS(dDtIni))
Do While SE1->(!Eof())
    @ lin, col SAY "teste"
```


Enddo

Nesse exemplo está faltando o comando para passar para o próximo registro dbkip(), ou seja, ele nunca será final de arquivo conforme a condição para sair do While.

Exemplo 2:

```
aCampos := {}  
Do while .T.  
    Aadd(aCampos, "Teste")  
Enddo
```

Nesse exemplo o sistema ficará rodando até alcançar o limite da variável tipo Array, até que isso aconteça o programa estará utilizando a memória do servidor e com isso diminuindo o desempenho de toda a empresa.

Exemplo 3:

```
dbSeek(xFilial("SE1")+DTOS(dDtIni))  
Do While SE1->(!Eof())  
    Reclock("SE5", .T.)  
Enddo
```

Nesse exemplo o sistema ficará rodando até acabar o espaço em disco no servidor, diminuindo o desempenho de toda a empresa e impedindo o uso por todos os usuários, até que o arquivo seja ajustado, apagando os registros incorretos.

Exercícios

1. Fazer Algoritmo que leia um vetor de 3 linhas e 5 colunas e imprima o seu conteúdo na tela, utilizando o que foi visto em vetores e estruturas de repetição.
2. Com base no Array informado mostrar na tela a media de cada aluno:

```
Local aArray := { { "Maria", 10, 7, 15, 31} ,;  
                  { "Jose ", 15, 16, 21, 33} , ;  
                  { "Petruncio", 8, 8, 8, 6} , ;  
                  { "Firmino", 15, 16, 21, 33} , ;  
                  { "Felizberto", 10, 17, 31, 25} }
```

3. Com base nesse mesmo array imprimir o nome informando mostrar na tela a media de cada aluno, conforme regra a seguir:

Nome do aluno seguida da palavra

14 **Aprovado** se a media for maior que 25

15 **Exame** se a media for entre 10 e 25

16 Reprovado se a media for menor que 10

4. Matheus é um homem de negócios e sempre viaja ao exterior e precisa controlar tudo que traz de lá. Sempre que ele traz mercadorias que ultrapassam R\$ 10.000,00, deve ser pago o imposto de 15%. Faça um algoritmo que leia o valor da mercadoria e grave na variável M o valor da mercadoria e se ultrapassar o valor, calcular o valor do imposto na variável I, caso não ultrapasse grave 0.
5. Calcule o Salário de um funcionário recebendo em uma variável o valor hora e em outra a quantidade de horas. Caso a qtde de horas ultrapasse a meta (180) acrescentar 2,00 por hora trabalhada e em seguida mostre na tela o salário a receber.
6. Desenvolva um fluxograma que:
 - Leia 4 (quatro) números;
 - Calcule o quadrado de cada um;
 - Se o valor resultante do quadrado do terceiro for ≥ 1000 , imprima-o e finalize;
 - Caso contrário, imprima os valores lidos e seus respectivos quadrados.
7. Elabore um algoritmo que dada a idade de um nadador classifique-o em uma das seguintes categorias:

Infantil A = 5 a 7 anos

Infantil B = 8 a 11 anos

Juvenil A = 12 a 13 anos

Juvenil B = 14 a 17 anos

Adultos = Maiores de 18 anos

Capítulo 09 – Development Studio

É o ambiente de desenvolvimento integrado do Advanced Protheus. É através dele que é feito o acesso ao repositório, incluindo e excluindo funções. O IDE é utilizado tanto pelos clientes quanto pelos programadores da Microsiga, com ele podemos compilar e depurar os programas. O IDE possui todos os recursos disponíveis nas melhores ferramentas de desenvolvimento do mercado.

Podemos utilizar o IDE sem conexão ao Server, a conexão somente é feita durante atualização ou consulta de um repositório ou durante o processo de depuração.

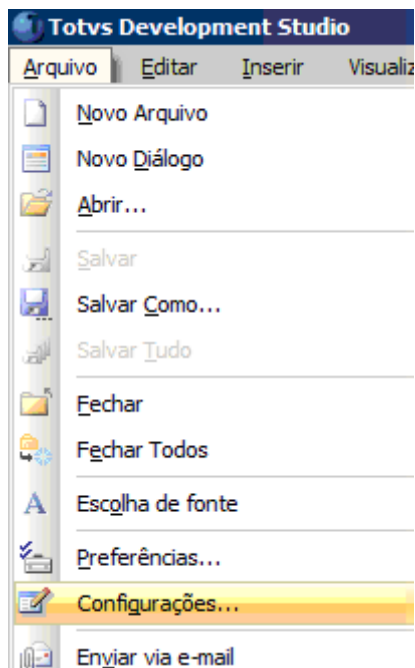
Os passos para o desenvolvimento de programas em AdvPL utilizando o IDE são:

- **Configuração de Ambiente** – Podemos configurar o ambiente que será atualizado na compilação e depuração de programas;
- **Criação do Projeto** – Criamos projetos como forma de organizar os programas criados de modo que qualquer outro analista possa entender o roteiro das customizações;
- **Criação do programa** – Através de edição podemos criar programas;
- **Compilação** – Compilamos os programas ou todo o projeto para enviarmos ao repositório o que foi mudado e para que o Protheus saiba o que queremos que faça;
- **Depuração** – Execução passo a passo das rotinas, permitindo que o analista veja o andamento do programa e o conteúdo das variáveis, bem como as tabelas abertas;

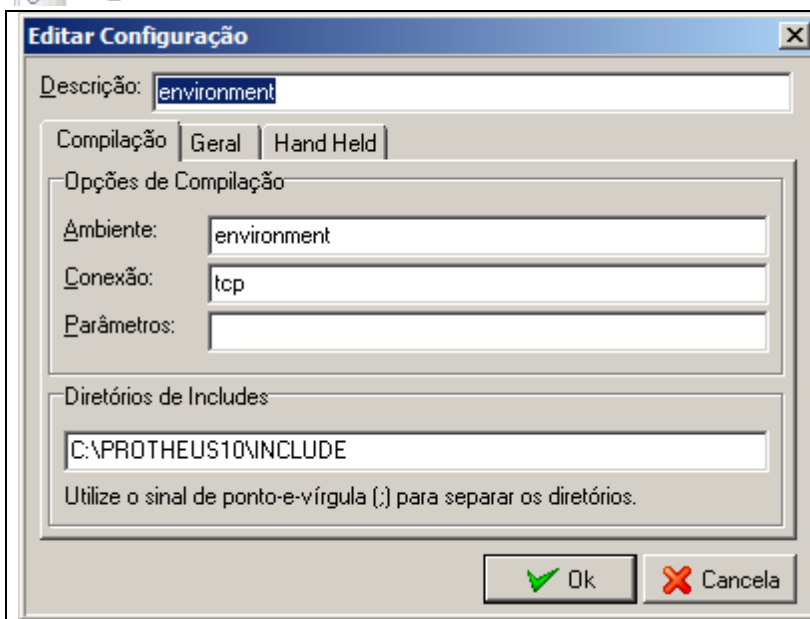
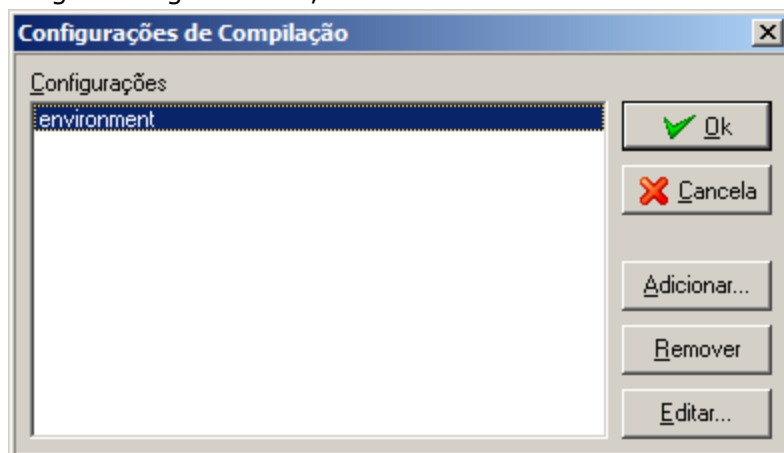
Configuração de Ambiente

Antes de começarmos, precisamos primeiro configurar o ambiente em que queremos trabalhar, isso é feito da seguinte forma:

Entramos no IDE e escolhemos a opção “Arquivo” e em seguida “Configurações”.



Surgirá a seguinte tela, escolha adicionar:



Descrição: Nome sugestivo para o ambiente, geralmente colocamos o mesmo nome do ambiente no Server

Ambiente: Nome do ambiente no server.

Conexão: nome da conexão utilizada, geralmente é TCP.

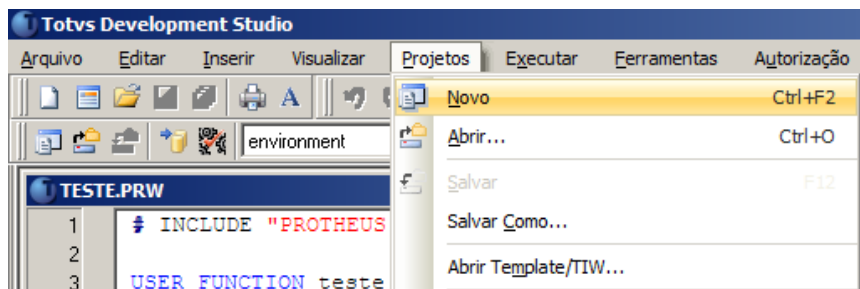
Parâmetros: Parâmetros utilizados (-m, -q, -a, -e) entre outros.

Diretórios de Includes:

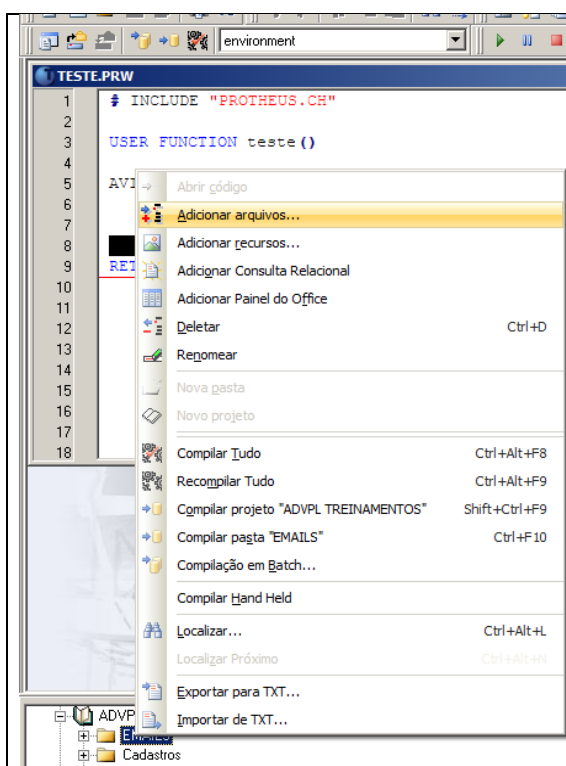
Caminho onde se encontram os includes, (*.ch), separados por ponto e virgula (;)

Criação de Projetos

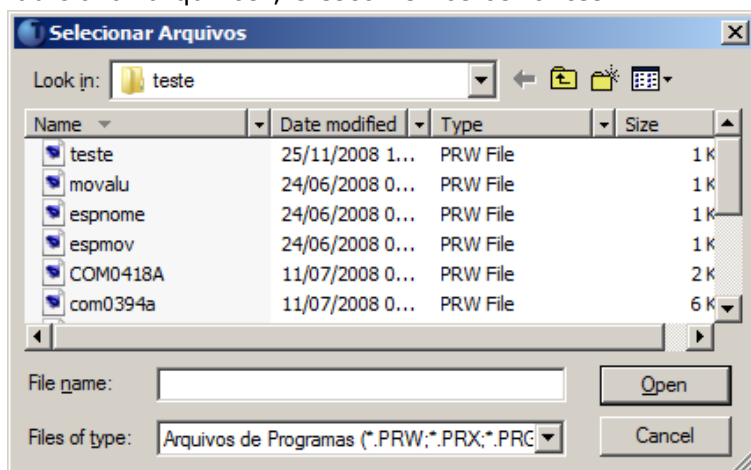
Após a configuração precisamos criar o projeto que conterá toda a organização dos programas envolvidos naquele projeto.



Assim que criamos um novo projeto podemos inserir programas nele, na estrutura em que achamos importante.




Para inserirmos arquivos no projeto, posicionamos o mouse na pasta em que queremos inserir e pressionamos o botão direito do mouse e escolhemos "adicionar arquivos", e escolhemos os fontes.



Utilizando assistentes de Programas

Editamos o programa no IDE, incluindo novos ou alterando um já existente. Podemos começar um programa do zero, mas também podemos usar o assistente de código. Clique em ferramentas e Assistente de código, surgira a seguinte tela, altamente intuitiva:

Assistente de Geração de Código



Tipo de Código

☒ Relatório

☐ Cadastro


☐ Geração de Arquivo Texto

☐ Importação de Arquivo Texto

☐ Processamento Genérico

Cancelar << Voltar Avançar >> Finalizar

Assistente de Geração de Código



Relatório - Arquivo Principal

☒ Arquivo Padrão

☐ Arquivo Específico

Alias:

☒ Utiliza índice padrão

☐ Utiliza índice temporário

Ordem:

Cancelar << Voltar Avançar >> Finalizar

Assistente de Geração de Código



Relatório - Parâmetros

Chave:

Cancelar << Voltar Avançar >> Finalizar

Assistente de Geração de Código



Relatório - Opções

Título:

Cabec. 1:

Cabec. 2:

☒ Normal Tamanho: 080 Colun

☐ Comprimido

Default Ordem Habilitações

Cancelar << Voltar Avançar >> Finalizar

Assistente de Geração de Código



Relatório - Opções

Ordem:



Default Ordem Habilitações

Cancelar << Voltar Avançar >> Finalizar

Assistente de Geração de Código



Relatório - Opções

☒ Mudança Impressão Comprimida/Normal

☒ Dicionário de Campos

☒ Filtro do Usuário

Default Ordem Habilitações

Cancelar << Voltar Avançar >> Finalizar

Agora que já temos o ambiente, projeto e os programas precisamos enviá-lo para o Protheus, para isso precisamos compilar o programa, e assim o programa será inserido no repositório e pode ser interpretado pelo Protheus.

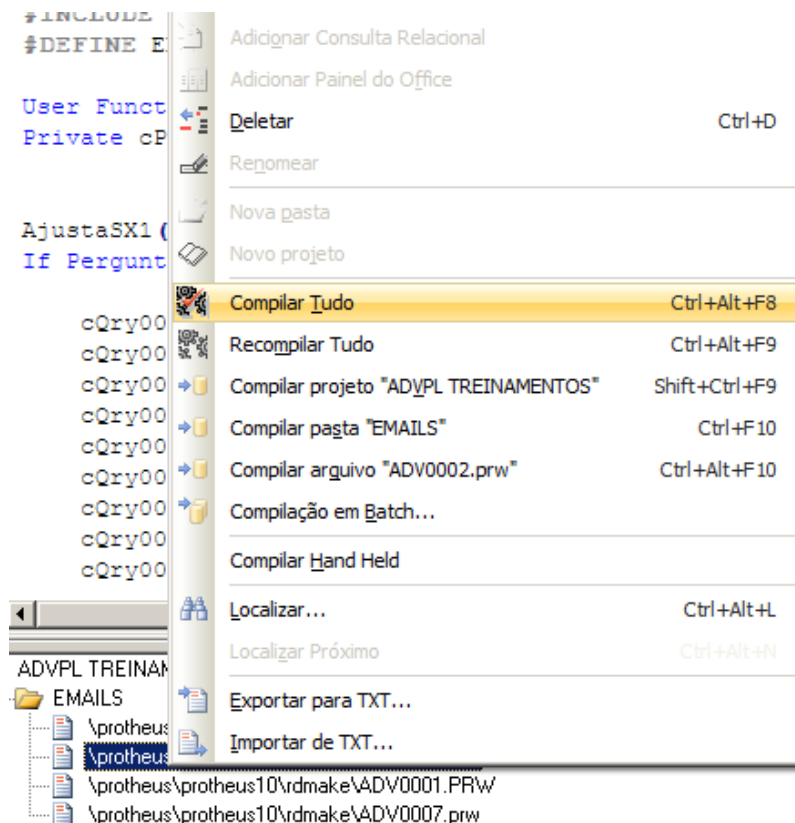
Compilação

Agora que já temos o ambiente, projeto e os programas precisamos enviá-lo para o protheus, para isso precisamos compilar o programa, e assim o programa será inserido no repositório e pode ser interpretado pelo Protheus.

Na hora de compilar, podemos escolher:

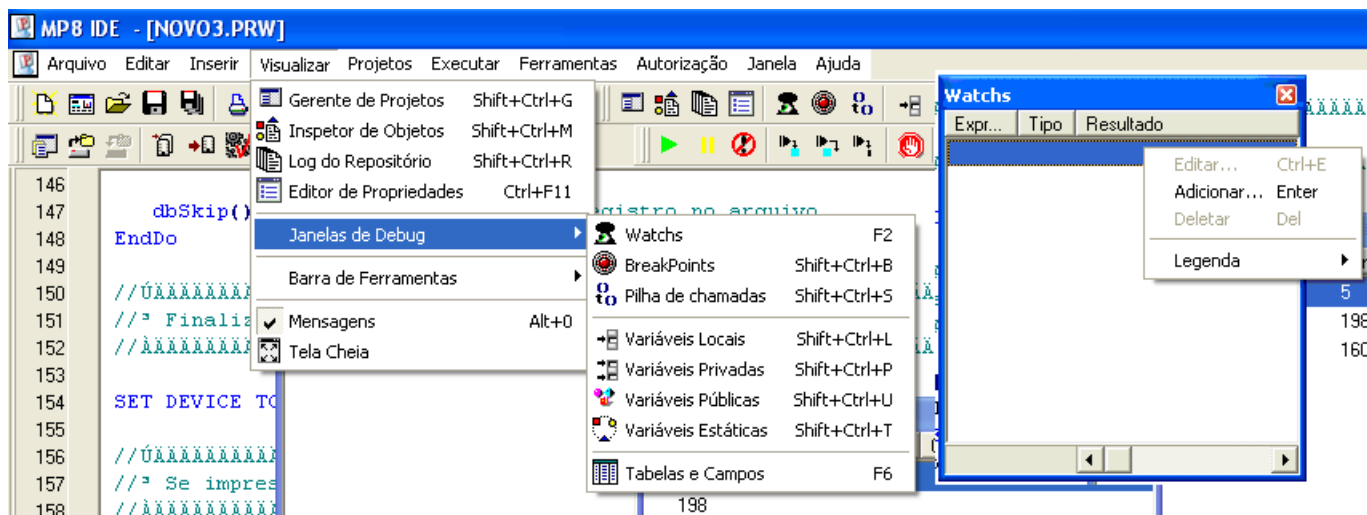
- Compilar tudo - todo o projeto, somente os que houve mudança desde a ultima compilação;
- Recompilar tudo – todo o projeto, sem exceção.
- Somente um determinado projeto
- Somente os programas dentro de uma pasta.
- Apenas um programa.

Para compilarmos basta clicar com o botão direito do mouse no projeto, surgirá a tela:



Nessa tela, o programa pararia na linha 160, quando chegasse no programa novo3.prw. Podemos visualizar o conteúdo das variáveis e campos nesse exato momento de algumas formas:

- Watches – Adicionamos as variáveis ou campos que queremos visualizar, e em momento de execução aparecerá o seu conteúdo, que são alteradas de acordo com o processamento.



Podemos ver também diretamente os tipos de variáveis, ou tabelas, escolhendo as opções:

- Variáveis Locais, Privadas, Publicas, estáticas ou tabelas e campos.

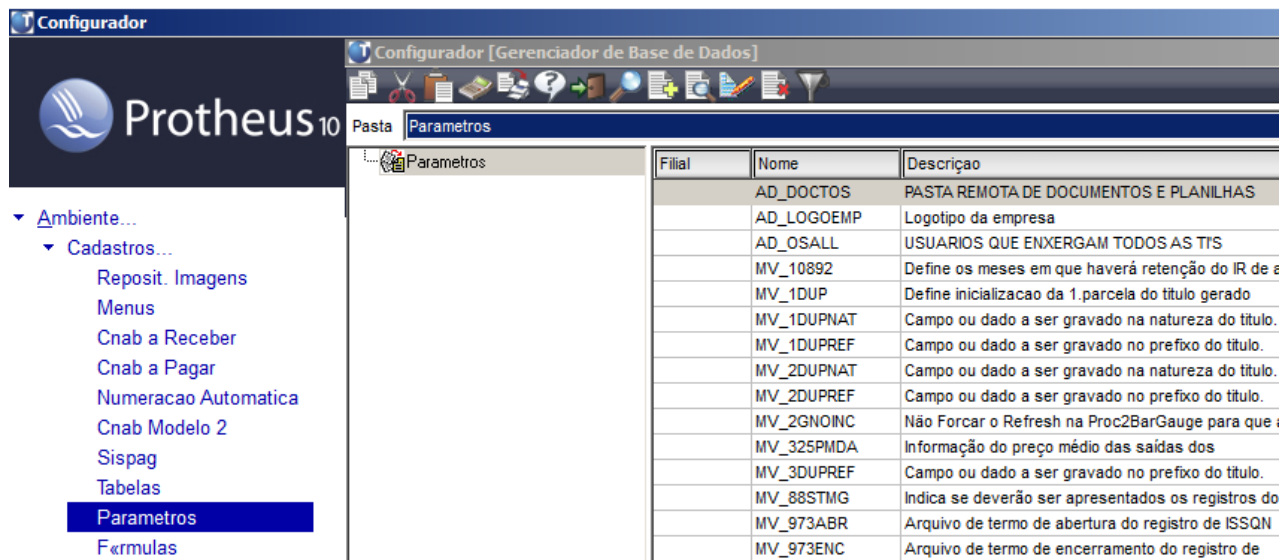
Capítulo 10 – Customização

Chamamos de customização tudo aquilo que fazemos no Protheus que foge do padrão, quando o padrão do Sistema não atende. Podemos customizar diversas situações:

- Parâmetros,
- Tabelas,
- Perguntas,
- Fórmulas - Expressões em AdvPL / User Function,
- Lançamentos Padrões via Expressões em AdvPL / User Function,
- Validações,
- Gatilhos,
- Campos de Arquivos,
- User Function via menu,
- Pontos de Entrada,
- Dicionário de Dados Ativo,
- SigarPM,
- Crystal,
- Integração Office (Word , Excel)

Parâmetros

Podemos criar parâmetros específicos para utilizar durante o processamento no sistema. Os parâmetros podem ser criados pelo configurador.



Filial	Nome	Descrição
	AD_DOCTOS	PASTA REMOTA DE DOCUMENTOS E PLANILHAS
	AD_LOGOEMP	Logotipo da empresa
	AD_OSALL	USUARIOS QUE ENXERGAM TODOS AS TTS
	MV_10892	Define os meses em que haverá retenção do IR de e
	MV_1DUP	Define inicializacao da 1.parcela do titulo gerado
	MV_1DUPNAT	Campo ou dado a ser gravado na natureza do titulo.
	MV_1DUPREF	Campo ou dado a ser gravado no prefixo do titulo.
	MV_2DUPNAT	Campo ou dado a ser gravado na natureza do titulo.
	MV_2DUPREF	Campo ou dado a ser gravado no prefixo do titulo.
	MV_2GNOINC	Não Forçar o Refresh na Proc2BarGauge para que i
	MV_325PMDA	Informação do preço médio das saídas dos
	MV_3DUPREF	Campo ou dado a ser gravado no prefixo do titulo.
	MV_88STMG	Indica se deverão ser apresentados os registros do
	MV_973ABR	Arquivo de termo de abertura do registro de ISSQN
	MV_973ENC	Arquivo de termo de encerramento do registro de

É importante estabelecermos um padrão na criação de novos parâmetros, para não serem confundidos com parâmetros padrões, apesar do sistema gravar o campo X6_PROPRI com

“U”, pode ocorrer uma coincidência na criação de parâmetros com nome igual a um parâmetro criado numa versão seguinte. Uma sugestão é colocar as iniciais da empresa seguida de números.

Existem funções que nos permite manipular os parâmetros.

GETMV - Retorna o conteúdo de um parâmetro cadastrado no SX6.

Sintaxe

GETMV(*cNomPar*, [*IRetPar*], [*uRet*])

Argumento	Obrig	po	Descrição
<i>cNomPar</i>	Sim	C	Nome do parâmetro a ser pesquisado
<i>IRetPar</i>	Não	L	Define retorno do parâmetro. Default .F. .F. - Retorna o conteúdo do parâmetro. (retorno de acordo com o tipo do parâmetro). .T. - Retorna se o parâmetro existe. (retorno lógico)
<i>uRet</i>	Não	U	Valor que deve ser retornado se o parâmetro solicitado não existir. O valor desse parâmetro pode ser caracter, numérico, lógico ou data.

Exemplo: MV_AD_0001 – Códigos de usuários autorizados e enxergar determinada rotina.

cADEmail := **GETMV("MV_AD_0001")** // Retorna o conteúdo do parametro

GETMV("MV_AD_0001", .T.) // Retorna .T. se o parâmetro existir e .F. se não existir.

// Retorna o conteúdo do parâmetro, caso não exista, retorna o "email@empresa.com.br"

cADEmail:= **GETMV(" MV_AD_0001", , "email@empresa.com.br")**

PUTMV – Grava informação no parâmetro no SX6.

Sintaxe

PUTMV(*cNomPar*, *cConteudo*)

Argumento	Obrig	po	Descrição
<i>cNomPar</i>	Sim	C	Nome do parâmetro a ser pesquisado
<i>cConteudo</i>	Sim	U	Conteúdo a ser gravado no parâmetro, deve ter o tipo esperado no parâmetro.

Exemplo: MV_AD_0001 – Códigos de usuários autorizados e enxergar determinada rotina.

PUTMV("MV_AD_0001", "email1@advpl.com.br")

PUTMVFIL – Grava informação no parâmetro no SX6 de acordo com a filial.

Sintaxe

PUTMVFIL(*cNomPar*, *cConteudo*, *cFil*)

Argumento	Obrig	po	Descrição
<i>cNomPar</i>	Sim	C	Nome do parâmetro a ser pesquisado
<i>cConteudo</i>	Sim	U	Conteúdo a ser gravado no parâmetro, deve ter o tipo esperado no parâmetro.
<i>cFil</i>	Não	C	Filial do parâmetro.

Exemplo: MV_AD_0001 – Códigos de usuários autorizados e enxergar determinada rotina.

PUTMVFIL("MV_AD_0001" , "email1@advpl.com.br" ,"01")

SUPERGETMV - Retorna o conteúdo de um parâmetro cadastrado no SX6.

Sintaxe

SUPERGETMV(cNomPar, [lHelp], [uRet] , [cFil])

Argumento	Obrig	po	Descrição
cNomPar	Sim	C	Nome do parâmetro a ser pesquisado
lHelp	Não	L	Define retorno do parâmetro. Default .F. .F. - Retorna o conteúdo do parâmetro. (retorno de acordo com o tipo do parâmetro). .T. - Retorna se o parâmetro existe. (retorno lógico)
uRet	Não	U	Valor que deve ser retornado se o parâmetro solicitado não existir. O valor desse parâmetro pode ser caracter, numérico, lógico ou data.
cFil	Não	C	Retorna parâmetro de acordo com a Filial, Default Filial atual. Se houver parâmetro para a filial, ela será priorizada.

Exemplo: MV_AD_0001 – Códigos de usuários autorizados e enxergar determinada rotina.

cADEmail := **SUPERGETMV("MV_AD_0001")** // Retorna o conteúdo do parametro

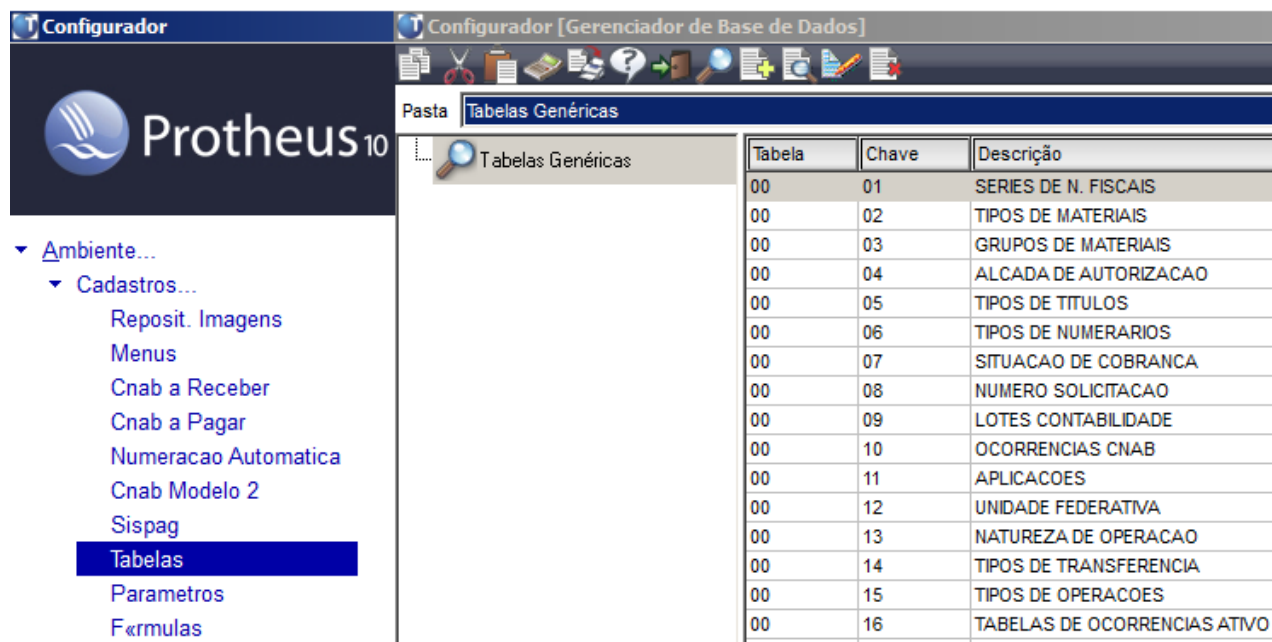
SUPERGETMV ("MV_AD_0001", .T.) // Retorna .T. se o parâmetro existir e .F. se não existir.

// Retorna o conteúdo do parâmetro da filial "01", caso não exista, retorna o "email@empresa.com.br"

cADEmail:= **SUPERGETMV (" MV_AD_0001", , email@empresa.com.br, "01")**

Tabelas

Podemos criar tabelas específicas para utilizar durante o processamento, fazer pesquisas e amarrações no sistema. As tabelas podem ser criadas pelo configurador.



A criação de tabelas deve ser feita quando temos alguma informação padronizada que contenha código e descrição, a microsiga reserva as tabelas iniciadas em Z para uso de clientes.

Existem funções que nos permite manipular as tabelas.

TABELA - Retorna o conteúdo de uma tabela cadastrada no SX5.

Sintaxe

TABELA(*cCodTab*, *cChave*, [*lHelp*])

Argumento	Obrig	po	Descrição
<i>cCodTab</i>	Sim	C	Código da Tabela a pesquisar
<i>cChave</i>	Sim	C	Chave para pesquisa da tabela
<i>lHelp</i>	Não	L	Define se mostra o help. Default .T. .T. – Mostra o help. .F. – Não mostra o help

Exemplo: Tabela "Z1" – Tipos de Móveis

`cADDescr := TABELA("00","07", .F.)` // Retorna a descrição da tabela '00' chave '07' que na figura será "TIPOS DE TITULOS"

X5DESCRI - Retorna a descrição no idioma atual da tabela posicionada no SX5.

Sintaxe

X5DESCRI()

Perguntas

Utilizado para criação de perguntas nos relatórios específicos, utilizando-se de recursos padrões do sistema, aproveitando funções prontas, com telas padrões.

PERGUNTE – Carrega e Monta a tela de perguntas cadastradas no SX1.

Sintaxe

PERGUNTE(*cGrupo*, *IMostra*)

Argumento	Obrig	Tipo	Descrição
<i>cGrupo</i>	Sim	C	Grupo de perguntas
<i>IMostra</i>	Sim	C	.T. – Carrega e mostra tela com perguntas, .F. apenas carrega respostas das perguntas

Expressões ADVPL

Podemos colocar qualquer expressão em ADVPL ou uma User Function em vários lugares no Protheus, dentre eles:

- Fórmulas (SM4)
- Lançamentos padrões (CT5)
- Validações (SX3 – X3_VLDUSER)
- Inicializador Padrão (SX3- X3_RELACAO)

Para executar essas instruções é utilizado o recurso de macrossubstituição.

Exemplo: Utilizando Cadastro de Formulas (M4_FORMULA)

U_FCRIAMENS() // Utilizando uma User Function

IF(SM4->M4_CODIGO < 'A01', 'NÃO ENVIAR', 'ENVIAR') //Utilizando uma expressão

"NÃO RECEBER APÓS AS 18 HRS " //Utilizando uma expressão sem condição

Para executar essa expressão é utilizado &M4_FORMULA.

Pontos de Entrada

São aberturas nas rotinas padrões do sistema que deixa o sistema flexível permitindo o desenvolvimento de processos específicos a partir de uma rotina padrão do sistema, de acordo com a necessidade do cliente e dos analistas de campo, permitindo maior abrangência nos diversos segmentos.

EXISTBLOCK() - verifica a existência de uma função de usuário compilada no repositório de objetos da aplicação Protheus. Esta função é normalmente utilizada nas rotinas padrões da aplicação Protheus para determinar a existência de um ponto de entrada e permitir sua posterior execução.

Sintaxe: EXISTBLOCK(cFun)

Argumento	Obrigat.	Tipo	Descrição
cFun	Sim	B	cFunção Nome da função que será avaliada
uparam	Não	U	Paarametros a enviar para a função

Retorno:- Lógico**.T. Existe a Função no repositório e .F. Não existe a função no repositório**

EXECBLOCK() - Executa uma função de usuário que esteja compilada no repositório. Esta função é normalmente utilizada pelas rotinas padrões da aplicação Protheus para executar pontos de entrada durante seu processamento.

- A função de usuário executada através da EXECBLOCK() não recebe parâmetros diretamente, sendo que estes estarão disponíveis em uma variável private denominada PARAMIXB.
- A variável PARAMIXB é o reflexo do parâmetro xParam, definido na execução da função EXECBLOCK(). Caso seja necessária a passagem de várias informações, as mesmas deverão ser definidas na forma de um array, tornando PARAMIXB um array também, a ser tratado na função de usuário que será executada.

Sintaxe: MExecAuto(cblock, , , uparam)

Argumento	Obrigat.	Tipo	Descrição
cblock	Sim	B	Nome da Função Padrão
uparam			

Exemplo

```
#include "protheus.ch"

User Function VerFun()

    IF EXISTBLOCK("MT100GRV")
        EXECBLOCK("MT100GRV",.F.,.F.,aParam)
    ENDIF

Return .t.
```

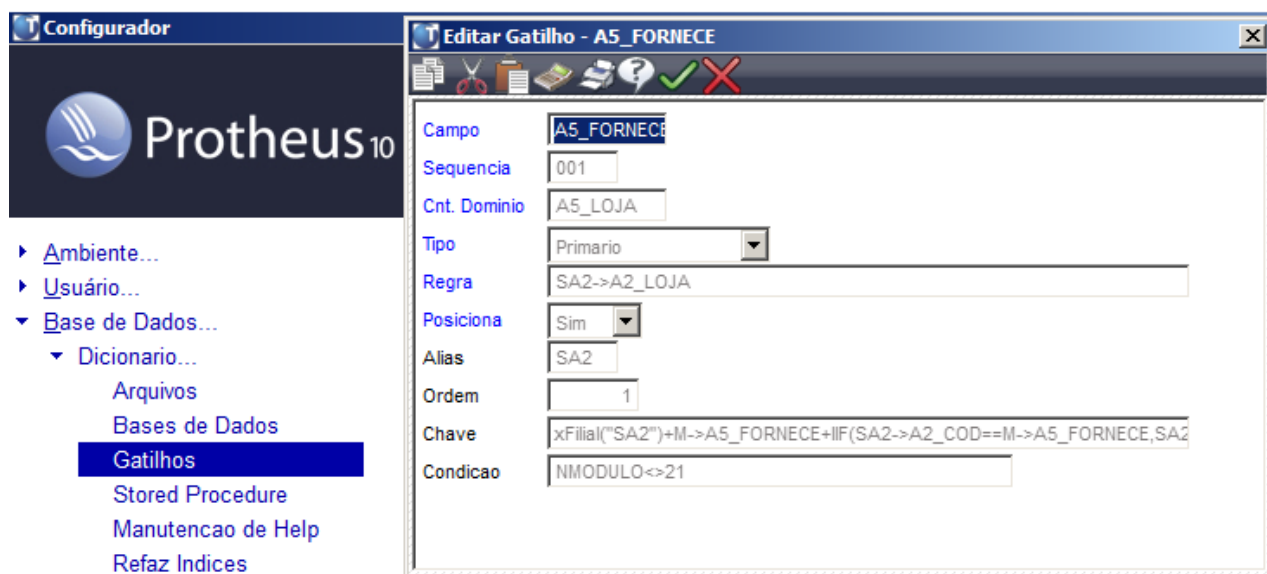

Dicionário de Dados Ativos

O Protheus tem um dicionário ativo, que permite alterações de informações que não necessitam que o sistema seja compilado para que entre em produção, bastando apenas alterar a partir do dicionário, que são tabelas com campos e registros que são lidos em tempo de execução e que utiliza o recurso de macros substituição.

Gatilhos

Rotina que define os campos que devem preencher outros campos no momento em que o campo for preenchido.

Exemplo: Ao digitar o código do fornecedor deverá preencher automaticamente o campo com o nome do fornecedor:



Campo: Nome do campo que dispara o gatilho

Seqüência: Seqüência de execução do gatilho

Cnt Domínio: Campo que receberá o preenchimento automático

Regra: Conteúdo que contra domínio recebera

Posiciona: Determina se devera fazer um posicionamento antes de executar a regra

Alias: se Posiciona for igual a "sim" , então deverá ser informado qual o alias a Posicionar

Ordem: Determina a ordem de pesquisa do Alias informado

Chave: chave a ser pesquisada no alias informado.

Condição: Se necessário, colocar a condição para execução do gatilho.

Capítulo 11 – Conceitos de Filiais

O Protheus permite a criação de empresas e filiais, permitindo tratamento de compartilhamento de informações entre as filiais em determinados cadastros.

O sistema permite controlar até 99 Empresas. Cada Empresa pode ter até 99 Filiais:

Exemplo:

Empresa 01 Filial 01..99

..

Empresa 99 Filial 01..99

Para cada empresa existe um jogo de Arquivos de dicionários e de tabelas também.

Sxxnn0 -> xx – Prefixo do arquivo e nn – Código da empresa

Exemplo:

Dicionário-> SX1010, SX2010, SX3010, etc

Tabelas -> SA1010, SA2010, SB1010, DA1010, etc

As filiais ficam dentro do arquivo, todo arquivo deve ter um campo filial.

xx_FILIAL -> xx – Prefixo do arquivo, quando a tabela começa com "S"

xxx_FILIAL -> xxx – Prefixo do Arquivo, quando a tabela não começa com "S"

Exemplo: A1_FILIAL, A2_FILIAL, B1_FILIAL, DA1_FILIAL.

Arquivos Compartilhados

Quando o arquivo está definido como compartilhado, no campo filial será gravado espaço em branco, essa definição é feita no dicionário de tabelas SX2 no campo X2_MODO = "C". Assim todas as filiais enxergarão todos os registros.

Exemplo:

A1_FILIAL	A1_COD	A1_NOME
	000001	CLIENTE A
	000002	CLIENTE B
	000003	CLIENTE C
	000004	CLIENTE D

Arquivos Exclusivos

Quando o arquivo está definido como exclusivo, no campo filial será gravada a filial, essa definição é feita no dicionário de tabelas SX2 no campo X2_MODO = "E", Assim somente a filial enxergará o registro.

Exemplo:

A1_FILIAL	A1_COD	A1_NOME
01	000001	CLIENTE A
01	000002	CLIENTE B
02	000003	CLIENTE C
01	000004	CLIENTE D

XFILIAL – Retorna a filial de um Alias específico.

Sintaxe

XFILIAL(*cAlias*)

Argumento	Obrig	Tipo	Descrição
<i>cAlias</i>	Sim	C	Alias do arquivo que deseja retornar a filial

Exemplo: xFilial("SA1")

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

A variável cFilAnt contém a filial que o usuário está no momento , e a variável cEmpant contém a empresa e a filial.

Versão 5.0 – 07/2010 – Todos direitos reservados

[illegible]

[illegible]

Static Function RunReport(Cabec1,Cabec2,Titulo,nLin)

Local nOrdem

```
(cString)->(dbSetOrder(1))
```

```
// SETREGUA -> Indica quantos registros serao processados para a regua
SetRegua(RecCount())
```

```
SB1->(dbGoTop())
While SB1->(!EOF())
```

```
// Verifica o cancelamento pelo usuario...
```

```
If IAbortPrint
```

@nLin,00 PSAY "*** CANCELADO PELO OPERADOR ***"

Exit

Endif

```
// Impressao do cabecalho do relatorio. . .
```

```
If nLin > 55 // Salto de Página. Neste caso o formulario tem 55 linhas...
```

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

$$n_{\text{Lin}} := 8$$

Endif

@nLin,00 PSAY SB1->B1 COD

```
@nLin,17 PSAY SB1->B1 DESC
```

```
@nLin,49 PSAY SB1->B1_TIPO
```

```
nLin := nLin + 1 // Avanca a linha de impressao
```

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

```
//Finaliza a execucao do relatorio...
SET DEVICE TO SCREEN
```

```
//Se impressao em disco, chama o gerenciador de impressao...
```

If aReturn[5] == 1

```
dbCommitAll()
```

SET PRINTER TO

OurSpool(wnrel)

Endif

MS_FLUSH()

Return

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

Argumento	Ident	Tipo	Descrição
wnRel	Local	C	Nome default do relatório em disco
cbCont	Local	N	Contador
Cabec1	Local	C	1ª linha do cabeçalho do relatório
Cabec2	Local	C	2ª linha do cabeçalho do relatório
Cabec3	Local	C	3ª linha do cabeçalho do relatório
Tamanho	Local	C	Tamanho do Relatório (P = 80 colunas, M = 132 colunas, G = 220 colunas)
cDesc1	Local	C	1ª linha da descrição do relatório
cDesc2	Local	C	2ª linha da descrição do relatório
cDesc3	Local	C	3ª linha da descrição do relatório
Limite	Local	N	Quantidade de colunas no relatório (80,132,220)
Titulo	Local	C	Título do Relatório
aReturn	Private	A	Matriz com as informações para a tela de configuração de impressão
Nomeprog	Private	C	Nome do programa do relatório
cString	Private	C	Alias do arquivo principal do relatório para o uso de filtro
Li	Private	N	Controle das linhas de impressão. Seu valor inicial é a qtde de linhas por página.
m_pag	Private	N	Controle do número de páginas do relatório
aOrd	Private	A	Matriz contendo as ordens para a impressão. Ex.: aOrd:={"Código","Descrição"}
nLastKey	Private	N	Utilizado para controlar o cancelamento da impressão do relatório
cPerg	Private	C	Nome da pergunta a ser exibida para o usuário
aLinha	Private	A	Matriz que contém informações para impressão de relatórios cadastrais

SETPRINT – Função de impressão padrão do Protheus

Sintaxe

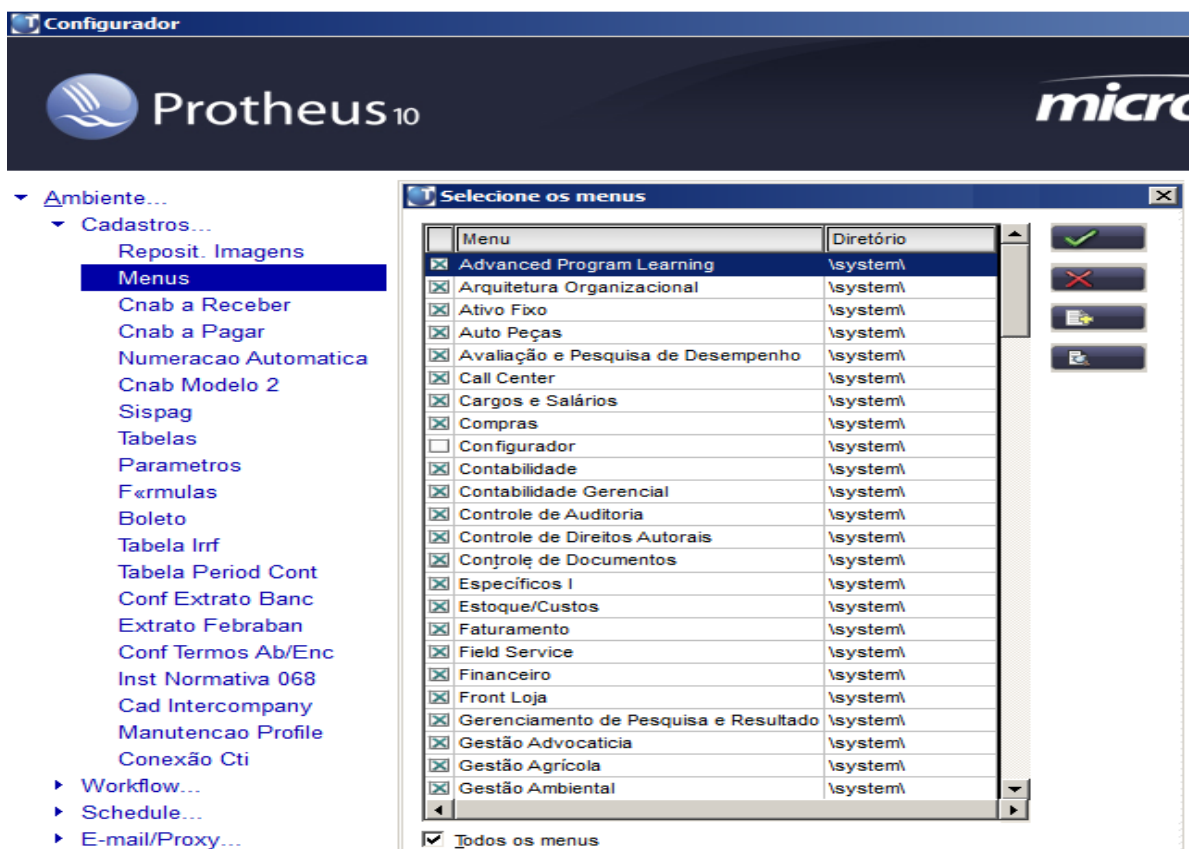
SetPrint(cAlias, cPrograma, [cPerg], [cTitulo], [cDesc1], [cDesc2], [cDesc3], [IDic], [aOrd], [ICompres], [cTam], [uPar1], IFiltro, [ICrystal], [cNomeDrv], [uPar2], [IServidor], [cPortaImpr]) -> caracter

Argumento	Ident	Tipo	Descrição
cAlias	Sim	C	Alias do arquivo a ser impresso.
cPrograma	Sim	C	Nome do arquivo a ser gerado em disco
cPerg	Não	C	Grupo de perguntas cadastrado no dicionário SX1.

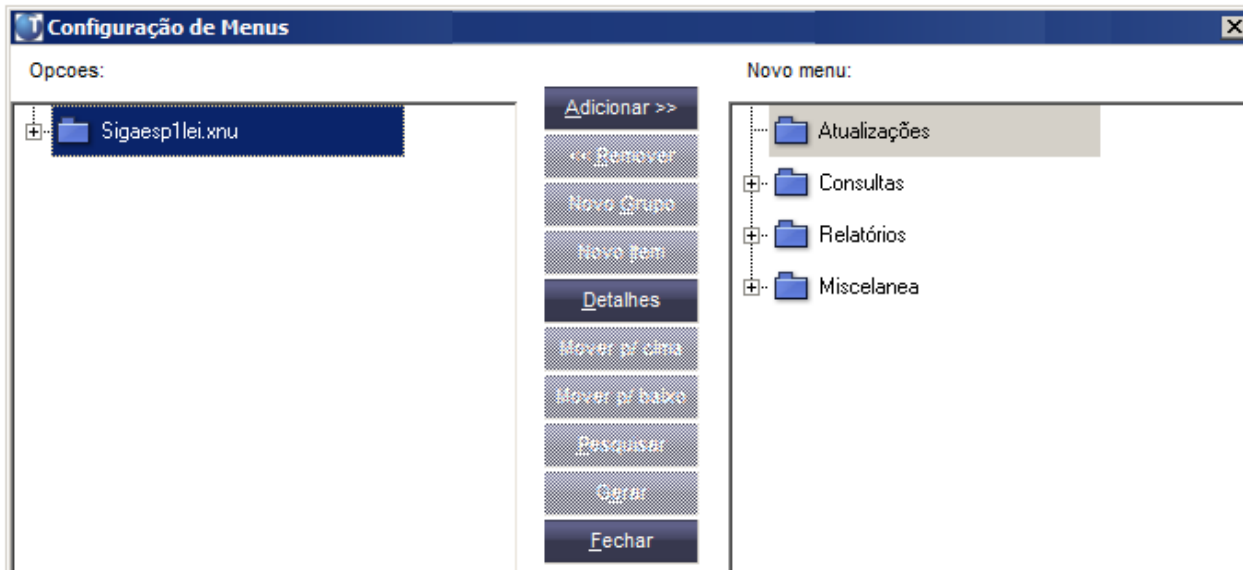
cTitulo	Não	C	Título do relatório
cDesc1	Não	C	Descrição do relatório.
cDesc2	Não	C	Continuação da descrição do relatório.
cDesc3	Não	C	Continuação da descrição do relatório.
IDic	Não	L	Permite a escolha dos campos a serem impressos.
aOrd	Não	A	Ordem(s) de impressão.
lCompres	Não	L	Se verdadeiro (.T.) habilita escolha o formato da impressão.
cTam	Não	C	Tamanho do relatório "P","M" ou "G".
uPar1	Não	U	Parâmetro reservado
IFiltro	Não	L	Se verdadeiro (.T.) permite a utilização do assistente de filtro.
ICrystal	Não	L	Se verdadeiro (.T.) permite integração com Crystal Report.
cNomeDrv	Não	C	Nome de um driver de impressão.
uPar2	Não	U	Parâmetro reservado.
IServidor	Não	L	Se verdadeiro (.T.) força impressão no servidor.
cPortaImpr	Não	C	Define uma porta de impressão padrão.

Incluindo Item no Menu

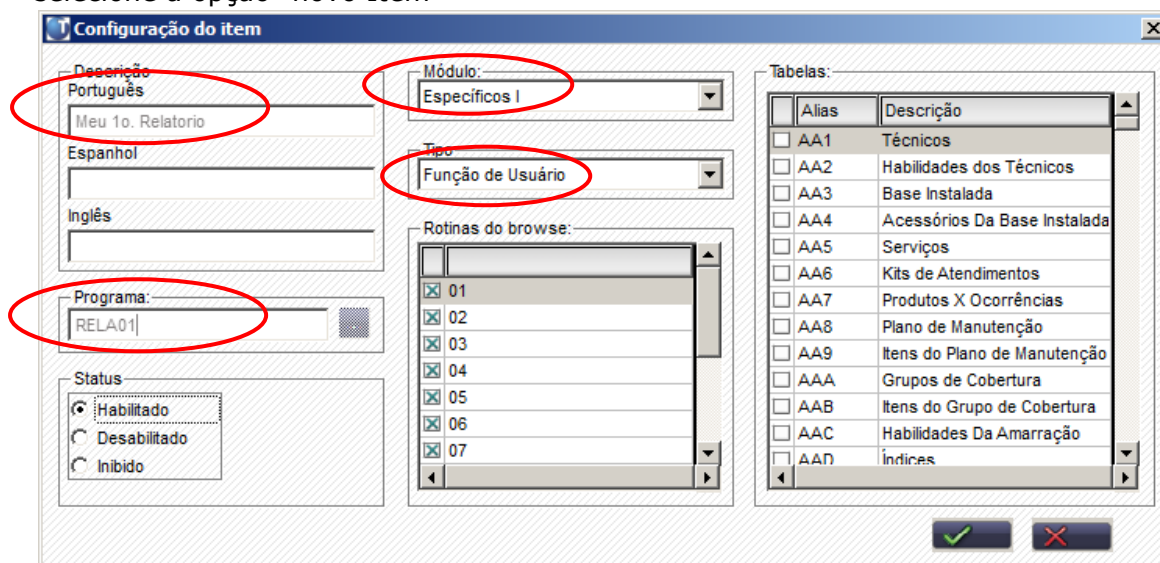
Entre no configurador, selecione Ambiente – Cadastros – Menus, surgira a seguinte tela:



Escolha o módulo que pretende alterar o menu e clique em ok



Clique em adicionar antes de fazer qualquer alteração para não sobrepor indevidamente o menu de produção, em seguida posicione no local em que deseja incluir o item no menu e selecione a opção "novo Item"



Descrição em Português: Descrição que você quer que apareça no menu;

Programa: Nome da User Function de seu programa;

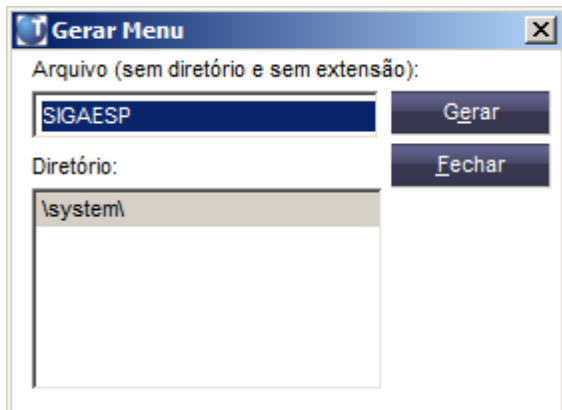
Módulo: escolha o módulo desejado;

Tipo: Escolha a opção adequada ao item no menu;

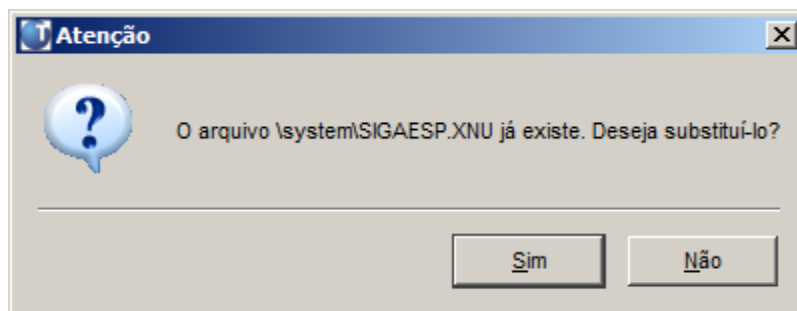
Rotina do Browse: Escolha os itens do browse que o usuário desse menu terá acesso;

Tabelas: Selecione as tabelas que esse programa precisará abrir.

Após ter preenchido todos os dados clique em OK e em seguida em gerar, surgirá a seguinte tela:



Coloque o nome do menu sem a extensão e clique em gerar, se o menu já existir aparecerá a seguinte tela:



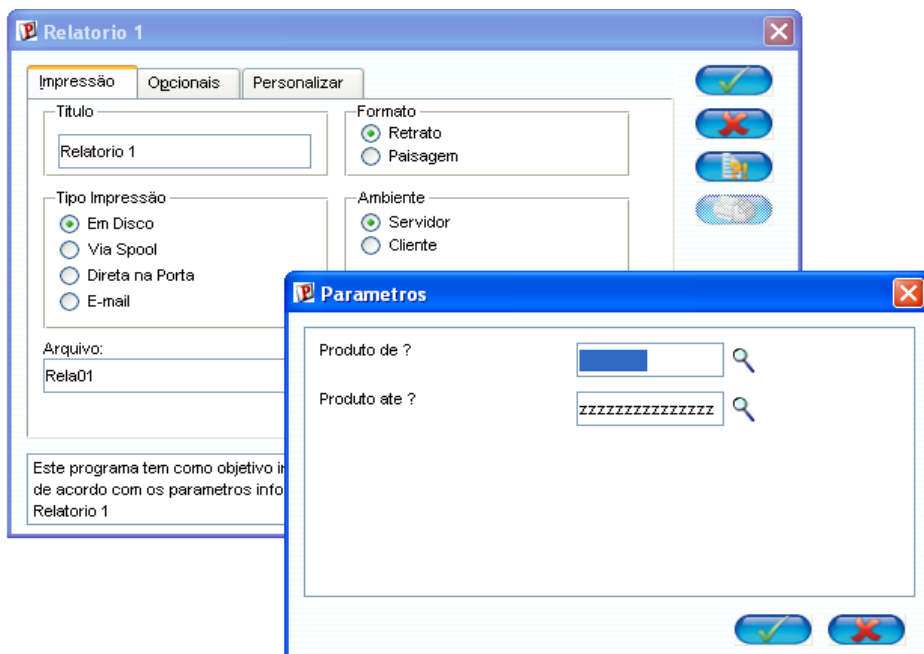
Clique em sim e em seguida clique em fechar.

Entre no módulo em que criou o item no menu e rode o programa para ver se tudo está ok.

Exercícios

1. Com base no exemplo de cadastro de produtos:
 - a. Faça um relatório que agrupe por tipo e demonstre quantos produtos de cada tipo existem.
 - b. Totalizar ao final do relatório
 - c. Criar pergunta para o relatório com o mesmo nome do programa: RELA01
Produto Inicial
Produto Final
 - d. Preparar o programa para entender tal filtro.

Veja a seguir como deverá ficar o relatório:



Relatorio 1

Impressão | Opcionais | Personalizar

Título: Relatorio 1

Formato: ☒ Retrato ☐ Paisagem

Tipo Impressão: ☒ Em Disco ☐ Via Spool ☐ Direta na Porta ☐ E-mail

Ambiente: ☒ Servidor ☐ Cliente

Arquivo: Rela01

Este programa tem como objetivo imprimir o relatório de acordo com os parâmetros informados no relatório 1.

Parametros

Produto de ?

Produto ate ?

coloque aqui o seu logo			Folha.: 1
SIGA /Rela01/v.MP8.11		Relatorio 1	DT.Ref.: 30/03/07
Hora.: 14:35:48			Emissao: 03/04/07
Codigo	Descricao	Tipo	
ATIVOS	ATIVOS DIVERSOS	GG	
TOTAL DE GG		1	
LUVA	LUVA PARA PROTECAO DAS MAOS	MC	
MANUTENCAO	ITEM PARA CONTROLE DE MANUTENC	MC	
TERCEIROS	ITEM PARA CONTROLE DE MANUTENC	MC	
TOTAL DE MC		3	
MOD0001	MAO DE OBRA FABRICACAO	MO	
MOD0002	MAO DE OBRA DO ACABAMENTO	MO	
TOTAL DE MO		2	
ARGOLA	ARGOLA PARA PRENDER CHAVES	MP	
ESCUDO	ESCUDO COM LOGOTIPO SIGA	MP	
MOSQUETAO	PRENDEDOR DO CHAVEIRO	MP	
REBITE	REBITE DE FERRO	MP	
SUPORTE	SUPORTE DE COURO	MP	
TIRANTE	TIRANTE DE COURO	MP	
TOTAL DE MP		6	
CHAVEIRO	CHAVEIRO DE BRINDE MICROSIGA	PA	
TOTAL DE PA		1	
CORPO	CORPO DE COURO	PI	
TOTAL DE PI		1	
TOTAL GERAL:		14	

Capítulo 13 – Funções Pré-Existentes

Por ser uma linguagem derivada do Clipper, podemos utilizar as diversas funções nativas da linguagem:

Manipulação de Stings

VAL - Converte valor em texto.

Sintaxe

VAL(cValor)

Argumento	Obrigat.	Tipo	Descrição
cValor	Sim	C	Conteúdo que se quer converter.

Exemplo:

cCodigo := "000015"

nNum := Val(cCodigo)+1

msginfo(nNum) // nNum terá o valor de 16

SUBSTR - Retorna um pedaço de um texto.

Sintaxe

SUBSTR(cConteudo, nValIni, nCount)

Argumento	Obrigat.	Tipo	Descrição
cConteudo	Sim	C	Conteúdo que se quer extrair uma parte
nValIni	Sim	N	Posição Inicial para extração
nCount	Sim	N	Qtde de caracteres a extrair a partir da Posição Inicial

Exemplo:

cCodigo := "Paralelepipedo"

cPedaco1 := Substr(cCodigo,5,4) // cPedaco1 terá o conteúdo "lele"

cPedaco2 := Substr(cCodigo,1,6) // cPedaco2 terá o conteúdo "parale"

cPedaco3 := Substr(cCodigo,9,5) // cPedaco3 terá o conteúdo "piped"

LEFT - Retorna o conteúdo de uma qtde de caracteres a esquerda

Sintaxe

LEFT(cConteudo, nCount)

Argumento	Obrigat.	Tipo	Descrição
<i>cConteudo</i>	Sim	C	Conteúdo que se quer extrair uma parte
<i>nCount</i>	Sim	N	Qtde de caracteres a extrair

Exemplo:

cCodigo := "Paralelepipedo"

cPedaco1 := Left(*cCodigo*,5) // *cPedaco1* terá o conteúdo "paral"

RIGHT – Retorna o conteúdo de uma qtde de caracteres a direita

Sintaxe

RIGHT(*cConteudo*, *nCount*)

Argumento	Obrigat.	Tipo	Descrição
<i>cConteudo</i>	Sim	C	Conteúdo que se quer extrair uma parte
<i>nCount</i>	Sim	N	Qtde de caracteres a extrair

Exemplo:

cCodigo := "Paralelepipedo"

cPedaco1 := Right(*cCodigo*,5) // *cPedaco1* terá o conteúdo "ipedo"

PADC – Centraliza um texto conforme a qtde de caracteres especificados.

Sintaxe

PADC(*cConteudo*, *nCount*)

Argumento	Obrigat.	Tipo	Descrição
<i>cConteudo</i>	Sim	C	Conteúdo que se quer centralizar
<i>nCount</i>	Sim	N	Qtde de caracteres a para centralizar

Exemplo:

cCodigo := "Parede"

cCodigo := Padc(*cCodigo*,14) // *cCodigo* terá o conteúdo " Parede "

PADR – preenche com espaços em branco a direita conforme a qtde de caracteres especificados.

Sintaxe

PADR(*cConteudo*, *nCount*)

Argumento	Obrigat.	Tipo	Descrição
<i>cConteudo</i>	Sim	C	Conteúdo que se quer alinhar
<i>nCount</i>	Sim	N	Qtde de caracteres a alinhar

Exemplo:

cCodigo := "Parede"

cCodigo := Padr(*cCodigo*,14) // *cCodigo* terá o conteúdo " Parede"

PADL – preenche com espaços em branco a esquerda conforme a qtde de caracteres especificados.

Sintaxe

PADL(cConteudo, nCount)

Argumento	Obrigat.	Tipo	Descrição
cConteudo	Sim	C	Conteúdo que se quer alinhar
nCount	Sim	N	Qtde de caracteres a alinhar

Exemplo:

cCodigo := "Parede"

cCodigo := Padl(cCodigo,14) // cCodigo terá o conteúdo "Parede "

ALLTRIM – Limpa espaços em branco iniciais e finais

Sintaxe

Alltrim(cConteudo)

Argumento	Obrigat.	Tipo	Descrição
cConteudo	Sim	C	Conteúdo que se quer limpar

Exemplo:

cCodigo := " Parede "

cCodigo := Alltrim(cCodigo) // cCodigo terá o conteúdo "Parede"

LTRIM – Limpa espaços em branco a esquerda.

Sintaxe

Ltrim(cConteudo)

Argumento	Obrigat.	Tipo	Descrição
cConteudo	Sim	C	Conteúdo que se quer alinhar

Exemplo:

cCodigo := " Parede "

cCodigo := Ltrim(cCodigo) // cCodigo terá o conteúdo "Parede "

RTRIM – Limpa espaços em branco a esquerda.

Sintaxe

Rtrim(cConteudo)

Argumento	Obrigat.	Tipo	Descrição
cConteudo	Sim	C	Conteúdo que se quer alinhar

Exemplo:

cCodigo := " Parede "

cCodigo := Rtrim(cCodigo) // cCodigo terá o conteúdo " Parede"

Manipulação de Dados

DATE – Retorna a data do sistema operacional

Sintaxe

DATE()

Exemplo:

dDataAtual := date() // dDataAtual terá o conteúdo da data de hoje

DTOS – Converte data em String

Sintaxe

DTOS(dData)

Argumento	Obrigat.	Tipo	Descrição
dData	Sim	D	Data a converter

Exemplo:

cData := dtos(date()) // cData terá o conteúdo 200612

DTOC – Converte data em Caracter

Sintaxe

DTOC(dData)

Argumento	Obrigat.	Tipo	Descrição
dData	Sim	D	Data a converter

Exemplo:

cData := dtoc(date()) // cData terá o conteúdo 31/12/2006

STOD – Converte String em data

Sintaxe

STOD(cData)

Argumento	Obrigat.	Tipo	Descrição
cData	Sim	C	Data a converter

Exemplo:

cData := "20061231"

cData := stod(date()) // cCodigo terá o conteúdo 31/12/06

STOC – Converte String em data

Sintaxe

STOC(cData)

Argumento	Obrigat.	Tipo	Descrição
cData	Sim	C	Data a converter

Exemplo:

cData := "31/12/06"

cData := stoc(date()) // cCodigo terá o conteúdo 31/12/06

MONTH – Retorna o Mes

Sintaxe

Month(dData)

Argumento	Obrigat.	Tipo	Descrição
dData	Sim	D	Data

Exemplo:

nMes := Month(date()) // nMes terá o conteúdo 12 para a data 31/12/06

DAY – Retorna o dia

Sintaxe

Day(dData)

Argumento	Obrigat.	Tipo	Descrição
dData	Sim	D	Data

Exemplo:

nDia := Day(date()) // nDia terá o conteúdo 31 para a data 31/12/06

YEAR – Retorna o Mes

Sintaxe

Year(dData)

Argumento	Obrigat.	Tipo	Descrição
dData	Sim	D	Data

Exemplo:

nAno := Year(date()) // nAno terá o conteúdo 2006 para a data 31/12/06

MESEXTENSO– Retorna o Mês por extenso em português

Sintaxe

Argumento	Obrigat.	Tipo	Descrição
dData	Sim	D	Data

Exemplo:

MesExtenso(dData)

cMes := MesExtenso(date()) // cMes terá o conteúdo Dezembro para a data 31/12/06

Manipulação de Números

STR – Converte numero em caracter

Sintaxe

Str(nConteudo)

Argumento	Obrigat.	Tipo	Descrição
nConteudo	Sim	N	Valor a converte

Exemplo:

nCodigo := 1

cCodigo := Str(nCodigo) // cCodigo terá o conteúdo " 1"

STRZERO – Converte numero em caracter,preenchendo com Zeros a esquerda.

Sintaxe

Strzero(nConteudo, nCount)

Argumento	Obrigat.	Tipo	Descrição
nConteudo	Sim	N	Valor a converte
nCount	Sim	N	Qtde de caracteres

Exemplo:

nCodigo := 1

cCodigo := Strzero(nCodigo, 6) // cCodigo terá o conteúdo "000001"

nCodigo := 158

cCodigo := Strzero(nCodigo, 6) // cCodigo terá o conteúdo "000158"

TRANSFORM – Converte numero em caracter utilizando máscara.

Sintaxe

Transform(nConteudo, cMask)

Argumento	Obrigat.	Tipo	Descrição
nConteudo	Sim	N	Valor a converte
CMask	Sim	N	Formato do caracter

Exemplo:

nCodigo := 1500

cCodigo := "Valor: " + Transform(nCodigo, "@E 99,999.99") // cCodigo terá o conteúdo (Valor: 1.500,00)

Manipulação de Vetores

AADD - Adiciona um novo elemento ao final do array.

Sintaxe

AADD(aAlvo, expValor)

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

Exemplos

```
aArray := {} // Resultado: aArray vazio
AADD(aArray, 5) // Resultado: { 5 }
AADD(aArray, 10) // Resultado: { 5, 10 }
AADD(aArray, { 12, 10 }) // Resultado: { 5, 10, { 12, 10 } }
```

ARRAY - Cria um array com dados não inicializados.

Sintaxe

ARRAY(nQtdElem1 , [nQtdElemn]...)

Argumento	Obrigat.	Tipo	Descrição
NQtdElem1	Sim	N	Quantidade de Elementos da 1ª dimensão do array.
[nQtdElemN]	Não	N	Quantidade de Elementos das demais dimensões do array.

Exemplos

```
aArray := ARRAY(5) ou aArray := { NIL, NIL, NIL, NIL, NIL }
aArray := ARRAY(3, 2) ou aArray := { {NIL, NIL}, {NIL, NIL}, {NIL, NIL} }
```

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

Sintaxe

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

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

Exemplo:

```
aArray := { "Tony", "Maria", "Sueli" }
nPos := ASCAN(aArray, "Maria") // Result: 2
nPos1 := ASCAN(aArray, "maria") // Result: 0
```

```
aArray5 := {}
AADD(aArray5, {"um", "dois"})
AADD(aArray5, {"tres", "quatro"})
AADD(aArray5, {"cinco", "seis"})
nPos3 := ASCAN(aArray5, { |x| x[2] == "quatro" }) // Result: 2
```

ASORT - Ordena um array.

Sintaxe

ASORT(aOrigem, [nInicio], [nQtde], [bOrdem])

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

Exemplo

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

```
aArray := { {"Maria", 14}, {"Joao", 23}, {"Arlete", 16} }
aArraycr := ASORT(aArray ,,, { |x, y| x[2]+x[1] < y[2] })
aArraydc := ASORT(aArray ,,, { |x, y| x[2] > y[2] })
aArraynom := ASORT(aArray ,,, { |x, y| x[1] < y[1] })
```

Capítulo 14 – Tratamento de Base de Dados

O Tratamento de base de dados em ADVPL é idêntico ao Clipper, utiliza-se as mesmas funções nativas para isso. No Protheus temos algumas funções que podemos utilizar para criação de arquivos e também criamos a partir do módulo configurador.

Criando Arquivos

CRIATRAB - Cria arquivo de trabalho, com nome temporário.

Sintaxe: CriaTrab([aCampos], [Exclus])

Argumento	Obrigat.	Tipo	Descrição
aCampos	Não	A	É o array com os campos da tabela a criar. Quando nulo apenas retorna um nome temporário e não cria a tabela.
Exclus	Sim	L	.T. cria tabela exclusiva .F. cria tabela compartilhada

Exemplo:

```
Local aCampos := { {'T_COD' , "C", 6, 0}, ;
                  {'T_NOME' , "C", 30, 0}, ;
                  {'T_ENDERECO' , "C", 30, 0}, ;
                  {'T_DATA' , "D", 8, 0}, ;
                  {'T_NUMERO' , "N", 17, 2} }
```

```
cArqTra := CriaTrab( aCampos, .T. ) //Cria o arquivo fisicamente em DBF no RothPath
```

```
dbUseArea(.t., cArqTra, "TMP", .f., .f.)
```

```
cArqTemp := CriaTrab( Nil, .F. ) //Somente gera um nome temporario
```

Criando Índices Temporários

INDREGUA - Cria índice temporário.

Sintaxe: INDREGUA(cAlias, cArqNtx, cIndCond, [cTipo], [cFiltro], cMensagem, [IMostra])

Argumento	Obrigat.	Tipo	Descrição
cAlias	Sim	C	Alias do arquivo que quer criar índice
cArqNtx	Sim	C	Nome físico do índice
cIndCond	Sim	C	Chave do índice
cTipo	Não	C	Tipo de índice "C" Crescente ou "D" Decrescente, se não informado o Default é "C" crescente.
cFiltro	Não	C	Filtro desejado

cMensagem	Sim	C	Mensagem a aparecer na tela
lMostra	Não	L	.T. Mostra Régua de progressão (default) .F. Não mostra a régua

Exemplo:

```
cArqTemp := CriaTrab( Nil, .F. ) //Somente gera um nome temporario
IndRegua( "SB1", cArqTemp, "B1_FILIAL+B1_GRUPO", , "B1_COD> 'D'",
"Selec.registros...", .T.)
//Deleta índice criado no final do programa
DbSelectArea( "SB1" ) //Selecionando a area
DbSetOrder( 1 ) //Posicionando na ordem de origem
fErase( cArqtemp + OrdBagExt() ) //Deletando arquivo de trabalho
```

Posicionamento de Registros

O posicionamento correto de registros é fundamental para a funcionalidade completa dos programas, é um grande causador de erros de lógica.

Algumas dicas para posicionamento de registros são :

- É sempre bom utilizar o Alias antes de comando de Base de Dados.
Exemplos:
SB1->(DBGOTOP()) – Posiciona no primeiro registro do arquivo SB1 de acordo com a ordem que esta selecionada no momento.
SB1->(DBGOBOTTOM()) – Posiciona no ultimo registro do arquivo SB1 de acordo com a ordem que esta selecionada no momento.
SB1->(DBSEEK(XFILIAL() + "000100")) - Busca em SB1 o registro que obedeça a chave estipulada de acordo com a ordem selecionada no momento.
- Ao executar um DBSEEK(), verificar se localizou o registro, exemplo:
If ! SB1->(dbSeek(xFilial("SB1")))
 // Não achei o registro
Endif
Sempre verifique a existência do registro, pois o programa deve prever que a base de dados pode não estar consistente, e um alerta ajuda a identificar estes casos. Em casos de relatórios, atentar-se para imprimir a mensagem de forma correta.
- Se for executada a função RECLOCK(cAlias, .F.), para alteração do registro atual, em um arquivo no estado de EOF() (caso falhe um DBSEEK()) será abortado o programa e gravado um arquivo texto de nome MSRLOCK.EOF que poderá ser usado para averiguações.

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

Exemplo :

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

- Ao criar uma função que irá desposicionar registros, use a função GETAREA() e RESTAREA(), para voltar tudo à posição original. Exemplo:
Dbselectarea("SD1")
aAreasd1 := Getarea() // Armazena o ambiente do arquivo SD1
SD1->(dbsetorder(3))
SD1->(dbseek(xfilial("SD1") + DTOS("01/03/01"), .T.))
Do While ! Eof() .And. D1_FILIAL == xfilial("SD1") .And. DTOS(D1_EMISSAO) <=
DTOS(mv_par02)
// Processamento
Dbskip()
Enddo
Restarea(aAreasd1) // Restaura o ambiente do arquivo SD1

- **Função Posicione**

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

Sintaxe:

Posicione(cAlias, nOrdem, cChave, cCampo)

Exemplo:

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

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

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

- **Função Existcpo**

Retorna se determinada chave existe ou não no arquivo.

Sintaxe :

ExistCpo(cAlias,cChave,nOrdem)

Exemplo :

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

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

Funções de Base de Dados

DBAPPEND – Inserir um registro em branco na tabela da área corrente.

Sintaxe: DBAPPEND()

DBSELECTAREA – Seleciona uma Área de Trabalho.

Sintaxe: DBSELECTAREA(cAlias)

Argumento	Obrigat.	Tipo	Descrição
CAlias	Sim	C ou N	Se numérico, seleciona área pelo número, se Caracter, seleciona a área pelo Apelido.

DBSETORDER – Seleciona um índice pela ordem.

Sintaxe: DBSETORDER(nOrd)

Argumento	Obrigat.	Tipo	Descrição
NOrd	Sim	N	Número do Índice

DBGOTOP – Posiciona o ponteiro no primeiro registro de acordo com a ordem corrente.

Sintaxe: DBGOTOP()

DBGOBOTTOM – Posiciona o ponteiro no ultimo registro de acordo com a ordem corrente.

Sintaxe: DBGOBOTTOM()

DBSEEK – Pesquisa na tabela corrente utilizando a chave da ordem corrente.

Sintaxe: DBSEEK(cChave, lEncontra)

Argumento	Obrigat.	Tipo	Descrição
CChave	Sim	C	Chave a ser procurada
lEncontra	Não	L	.T. – Posiciona no registro da chave, se não encontrar posiciona no registro mais próximo a chave. .F. – (Default) Posiciona no registro da chave, se não encontrar posiciona no fim do arquivo (EOF)

DBSKIP – Salta registro da tabela corrente.

Sintaxe: DBSKIP(nStep)

Argumento	Obrigat.	Tipo	Descrição
NStep	Nao	N	Default 1. Qtde de registros que será saltado. Se positivo avança registros, Se negativo retrocede registros.

DBFILTER – Retorna a expressão do filtro da tabela corrente.

Sintaxe: DBFILTER()

DBSETFILTER – Retorna a expressão do filtro da tabela corrente.

Sintaxe: DBSETFILTER(bFiltro, cFiltro)

Argumento	Obrigat.	Tipo	Descrição
BFiltro	Sim	B	Bloco com a expressão de filtro
CFiltro	Sim	C	Expressão do Filtro

DBCLEARFIL – Limpa o filtro da tabela corrente.

Sintaxe: DBCLEARFIL()

DBGOTO – Posiciona no registro especificado da tabela corrente.

Sintaxe: DBGOTO(nReg)

Argumento	Obrigat.	Tipo	Descrição
NReg	Sim	N	Numero do Registro

DBUSEAREA – Abre uma tabela e a deixa corrente.

Sintaxe: DBUSEAREA(INew, cDriver, cNomArq, cAlias, IShared, IReadOnly)

Argumento	Obrigat.	Tipo	Descrição
LNew	Não	L	.T. Abre em uma nova Area .F. (Default) Abre na área corrente, fechando a área anteriormente utilizada.
CDriver	Não	C	Driver da tabela a abrir. Default DBFCDX
cNomArq	Sim	C	Nome da tabela a abrir
CAlias	Sim	C	Apelido
LShared	Não	L	.T. Abre a tabela compartilhada .F. (Default) Abre a tabela exclusiva
IReadOnly	Não	L	.T. Abre a tabela somente leitura .F. (Default) Permite gravação na tabela

Exemplo:

dbUseArea(.T., "TOPCONN", "SX5020", "SX5A", .T., .T.)

dbSelectArea("SB1")

dbSetorder(1)

dbGotop()

cFiltro := "B1_TIPO > "MP"


```
cADFilAnt := SB1->(dbFilter())
SB1->(dbSetFilter({||&cFiltro},cFiltro))
While SB1->(!EOF())
    If !dbSeek(xFilial("SB1") + SB1->B1_COD)
        SB5->(dbAppend())
    Endif
    SB5->B5_FILIAL := xFilial("SB1")
    SB5->B5_COD := SB1->B1_COD
    SB1->(MsUnlock())
    SB1->(dbSkip())
Enddo
If Empty(cADFilAnt)
    SB1->(dbClearFil())
Else
    SB1->(dbSetFilter({||&cADFilant},cADFilant))
Endif
```

Capítulo 15 – Blocos de Códigos

Definição

O tipo de dado chamado bloco de código permite que o código compilado seja tratado como um dado. Pode-se considerar um bloco de código como uma “**função sem nome**” atribuída a uma variável. E como o AdvPL trata os blocos de código como valores, eles podem ser passados como parâmetros e armazenados em variáveis.

As operações que podem atuar em um bloco de código são as de atribuição e avaliação. A atribuição de um bloco de código é feita geralmente a uma variável, mas eles também podem ser um elemento de um vetor ou passados como parâmetros. A avaliação de um bloco de código pode ser realizada sobre um valor, para cada elemento de um vetor, ou um bloco de código para avaliar cada registro de uma base de dados.

Sintaxe:

$uRet := \{ | <argumentos> | <expressões> \}$

Argumento	Obrigat.	Tipo	Descrição
argumentos	Não	Diversos	Lista de argumentos que o bloco receberá, cada argumento deve ser separado por virgula.
expressões	Sim	Diversos	Lista de expressões que o bloco executará, cada instrução deverá ser separada por virgula
URet			Retorno será o resultado da ultima expressão da lista de expressões.

Exemplo:

$bBloco := \{ | nVar1, cVar1 | fTeste(nVar1, cVar1) \}$

Como um bloco de código é como uma função sem nome, veja o bloco e sua representação equivalent abaixo:

Bloco	Equivalente
$bTest := \{ nVar nVar * 5 \}$	Function SemNome(nVar) Return(nVar * 5)

Funções de Bloco

EVAL - Executa um bloco de código qualquer, com passagem de parâmetros.

Sintaxe: `uRet := Eval(bVar,Par1,Par2...Parx)`

Argumento	Obrigat.	Tipo	Descrição
BVar	Sim	Bloco	Bloco a ser executado
Par1,Par2..Parx	Não	Diversos	Lista de argumentos que o bloco informado necessita receber.
URet			Retorno será o resultado da ultima expressão da lista de expressões.

Exemplo:

`bBloco := {|nvar| nVar * 10}`

`nRet := Eval(bBloco,9) // resulta 90`

ou

`bVar := {| | N1 := 6, N2 := 20, N3 := N1 * N2}`

`nVar := Eval(bVar) // Resulta 120`

Equivalente a

`N1:= 6`

`N2:=20`

`N3:= N1 * N2`

DBEVAL - Executa um bloco de código enquanto obedecer a condição.

Sintaxe: `dbEval(bVar,bFor,bWhile)`

Argumento	Obrigat.	Tipo	Descrição
BVar	Sim	Bloco	Bloco a ser executado
BFor	Não	Bloco	Lista de argumentos que o bloco informado necessita receber.
BWhile	Não	Bloco	Retorno será o resultado da ultima expressão da lista de expressões.

Exemplo:

`bBloco := {| | aAdd(aTabela,{X5_CHAVE,Capital(X5_DESCRI)})}`

`bWhile := {| | X5_TABELA==cTabela}`

`dbEval(bBloco,,bWhile)`

Equivalente a

`While !Eof() .And. X5_TABELA==cTabela`

`aAdd(aTabela,{X5_CHAVE,Capital(X5_DESCRI)})`

`dbSkip()`

`End`

AEVAL - Executa um bloco de código, passando um array como parâmetro, a função percorre o array de acordo com os argumentos de inicio e quantidade de vezes do array.
Sintaxe: aEval(aArray,bBloco,nInicio,nCount)

Argumento	Obrigat.	Tipo	Descrição
AArray	Sim	Array	Array eu deverá ser percorrido
BBloco	sim	Bloco	Bloco com instruções a executar
NInicio	Não	Numerico	Elemento que o bloco começará a percorrer, se não informado o default será 1
NCount	Não	Numerico	Numero de elementos que o bloco deve percorrer, se não informado, o default é ir até o final do array.

Exemplo:

```
aEstrutura := (cAlias)->(dbStruct()) //{cCampo,cTipo",nTam,nDec}
bAcao := { |x| Msginfo(aEstrutura[x][1]) }
aEval(aEstrutura, bAcao)
```

Equivalente a

```
aEstrutura := (cAlias)->(dbStruct()) //{cCampo,cTipo",nTam,nDec}
For I:=1 to len(aEstrutura)
    Msginfo(aEstrutura[I][1])
Next
```

A variável x foi criada para guardar o elemento da matriz que está sendo lido no momento, tornando possível o dinamismo na apresentação da mensagem.

Capítulo 16 – Funções Diversas

Funções de Ambientes

GetRemoteIniName() - Retorna o nome do arquivo de configuracao remote.

GetEnvServer() - Retorna o ambiente atual

GetTheme() - Retorna o tema escolhido.

GetDBExtension() - Retorna a extensão da Base (DBF, DTC, etc)

Conout(cMens) - Imprime cMens no console do Server

OrdBagExt() - Retorna a extensão do Indice(CDX, IDX, NTX, etc)

Funname(n)- Retorna a função da pilha, onde n é a posição na pilha.

Exemplo:

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

```
User Function FuncServ()
```

```
    cTemProc := "Menu: " + CARQMNU + CRLF //Nome do menu
```

```
    cTemProc += "Modulo: " + cModulo+ CRLF // Modulo atual
```

```
    cTemProc += "Ambiente: " + Getenvserver() + CRLF //Ambiente atual
```

```
    cTemProc += "Usuario: " + cUserName + CRLF //Nome Usuario
```

```
    cTemProc += "Programa: " + funname() + CRLF //Funcao chamada Menu
```

```
    cTemProc += "Tema: " + GetTheme() + CRLF //Tema do usuario
```

```
    cTemProc += "Extensao Base Local: " + GetDBExtension() + CRLF //Extensão  
LocalFiles
```

```
    cTemProc += "Tipo Indice: " + OrdBagExt() + CRLF //Extensao de índices
```

```
    conout(cTemProc)
```

```
Return Nil
```

Funções de Servidor

GetsrvProfString() - Retorna o conteudo do parametro da seção

Sintaxe: GetSrvprofString(cParam, cDefault)

Argumento	Obrigat.	Tipo	Descrição
cParam	Sim	C	Parametro da seção no arquivo INI
cDefault	Sim	C	Conteúdo Padrão, caso não seja encontrado o parâmetro "cParam" na seção do INI.

GetFuncArr() – Retorna um array com as funções do repositório do ambiente corrente.

Sintaxe: aVar := GetFuncArr(cNomFun)

Argumento	Obrigat.	Tipo	Descrição
cNomFun	Sim	C	Nome da função do repositório, se informado "A", retorna todas as funções do repositório.

GetFuncPrm(cFun) – Retorna parâmetros da função especificada.

Sintaxe: : aVar := GetFuncPrm(cFun)

Argumento	Obrigat.	Tipo	Descrição
cFun	Sim	C	Nome da função do repositório.

Exemplo:

```
#include "protheus.ch"
User Function FuncServ()
Local aGetA := {}
Local aGetF := {}
Local cFun := ""
aGetA := GetFuncArr("A")
For I:=1 to Len(aGetA)
    cFun+= aGetA[I] + "() "
    aGetF := GetFuncPrm(aGetA[I])
    For X:=1 to Len(aGetF)
        cFun += If(X>1, ", ", "- ") + aGetA[X]
    Next
Next
cLocalArq := GetsrvProfString("STARTPATH", "C:\")
Memowrite(cLocalArq,cFun)
```

Return Nil

GetApoInfo() – Retorna um array com dados do programa informado dentro do repositório.

Sintaxe: aVar := GetApoInfo(cNomProg)

Argumento	Obrigat.	Tipo	Descrição
cNomProg	Sim	C	Nome do programa

Exemplo:

```
#include "protheus.ch"
User Function fDadosFun()
```

```
Local aApoInfo := GetApoInfo( "MATA410.PRX" )

Conout(aApoInfo[1]) //MATA410.PRX -> Nome do arquivo
Conout(aApoInfo[2]) //ADVPL      -> Codificada Em
Conout(aApoInfo[3]) //BUILD_FULL -> Quem Compilou, TOTVS ou Cliente
Conout(aApoInfo[4]) //23/01/2007 -> Data da última Modificação.

Return
```

Funções de Comunicação Server x Client

CpyS2T() - Copia arquivos do servidor para a máquina do cliente.

Sintaxe: CPYS2T(cArqOri, cArqDes, lComp)

Argumento	Obrigat.	Tipo	Descrição
cArqOri	Sim	C	Arquivo de Origem, visível a partir do rootpath do ambiente corrente
cArqDest	Sim	C	Arquivo de Destino, visível na máquina do cliente
lComp	Não	L	Define se compacta antes de enviar o arquivo. Default .T.

Exemplo:

```
#include "protheus.ch"
User Function CopiaArq()
Local cArqOri := "\SYSTEM\SIGAFIN.XNU"
Local cArqDes := "C:\TEMP"

If CPYS2T(cArqOri, cArqDes, .T.)
    MSGINFO("Arquivo " + cArqOri + " copiado com sucesso para " + cArqDes,
    "COPIAARQ")
Else
    MSGINFO("Arquivo " + cArqOri + " não foi copiado! ", "COPIAARQ")
EndIf

Return Nil
```

CpyT2S() - Copia arquivos da máquina do cliente para o servidor.

Sintaxe: CPYT2S(cArqOri, cArqDes, lComp)

Argumento	Obrigat.	Tipo	Descrição
cArqOri	Sim	C	Arquivo de Origem, visível na máquina do cliente
cArqDest	Sim	C	Arquivo de Destino, visível a partir do rootpath do ambiente corrente
lComp	Não	L	Define se compacta antes de enviar o arquivo. Default .T.

Exemplo:

```
#include "protheus.ch"
User Function CopiaArq()
Local cArqOri := "C:\TEMP\SIGAFIN.XNU"
Local cArqDes := "\SYSTEM "

If CPYT2S(cArqOri, cArqDes, .T.)
    MSGINFO("Arquivo " + cArqOri + " copiado com sucesso para " + cArqDes,
    "COPIAARQ")
Else
    MSGINFO("Arquivo " + cArqOri + " não foi copiado! ", "COPIAARQ")
EndIf

Return Nil
```

Funções de Servidor

RpcClearEnv() – Limpa o ambiente

RpcSetType() – Seleciona o Tipo o RPC

RpcSetEnv() – Prepara o Ambiente para a empresa definida

Sintaxe: RpcSetEnv(cADEmp,cADFil,cUser,cSenha,cADMod,,aTables)

Argumento	Obrigat.	Tipo	Descrição
cADEmp	Sim	C	Código da Empresa
cADFil	Sim	C	Código da Filial
cUser	Não	C	Usuário do Protheus
cSenha	Não	C	Senha do Protheus
cADMod	Não	C	Sigla do módulo
aTables	Não	A	Array com os alias necessários

Exemplo:

```
#include "protheus.ch"
User Function PrepEnv()

Local cUser:= "Administrador"
Local cSenha:="123456"
Local aEmps := {{"01","01"};{"02","01"};{"03","01"};{"99","01"}}

  For I:=1 to Len(aemps)
    RpcClearEnv()
    RpcSetType(3)
    RpcSetEnv(aEmps[I,1], aEmps[I,2],cUser,cSenha,"FAT",,{"SA1",
    "SA2","SA3"})
    Msginfo(SM0->M0_NOME, "PREPENV")
  Next

Return Nil
```

Pode ser feito de outra forma:

Exemplo:

```
#include "protheus.ch"
User Function PrepEnv()

Local cUser:= "Administrador"
Local cSenha:="123456"
Local aEmps := {{"01","01"};{"02","01"};{"03","01"};{"99","01"}}

  For I:=1 to Len(aemps)
    RpcClearEnv()
    RpcSetType(3)
    PREPARE ENVIRONMENT EMPRESA aEmps[I,1] FILIAL aEmps[I,2] USER cUser;
    PASSWORD cSenha MODULO 'FAT' TABLES "SA1", "SA2", "SA3"
    Msginfo(SM0->M0_NOME, "PREPENV")
  Next

Return Nil
```

Capítulo 17 – Tela de Padrão Microsiga

AxCadastro()

Tela de cadastro simples, com poucas opções de customização, a qual é composta de:

- Browse padrão para visualização das informações da base de dados, de acordo com o Dicionário de Dados.(X3_BROWSE = 'S')
- Funções padrões para visualização de registros simples, sem a opção de cabeçalho e itens.
 - axPesqui() – Tela de pesquisa de acordo com dicionario (SIX)
 - AxVisual() – Tela de visualização de acordo com o dicionario (SX3)
 - AxInclui()– Tela de inclusão de acordo com o dicionario (SX3)
 - AxAltera()– Tela de alteração de acordo com o dicionario (SX3)
 - AxDeleta()– Tela de exclusão de acordo com o dicionario (SX3)

Sintaxe: AxCadastro(cAlias, cTitulo, [cVldExc], [cVldAlt], [arotadic], [bpre], [bok], [aauto], [nopcauto],; [btts], [bnotts], [abuttons])

Argumento	Obrigat.	Tipo	Descrição
cAlias	Sim	C	Alias do arquivo
cTitulo	Sim	C	Titulo da Janela
cVldExc	Não	C	Nome de funcao para validacao da exclusao
cVldAlt	Não	C	Nome de funcao para validacao da tela
Arotadic	Não	A	Array de botões adicionais no arotina
Bpre	Não	B	CodeBlock Executado antes da Interface
Bok	Não	B	CodeBlock Executado na validacao da Interface
Aauto	Não	A	Array com dados para execução de rotina automática
Nopcauto	Não	N	Opção para execução de rotina automática
Btts	Não	B	CodeBlock Executado dentro da transação
Bnotts	Não	B	CodeBlock Executado fora da transação
Abuttons	Não	A	CodeBlock com botões adicionais na enchoicebar

Retorno:- Lógico

.T. Clique no botão OK da tela

.F. Clique o botão Cancela da tela

Exemplo:

```
#include "protheus.ch"
User Function CadSZ9()
```

```
Local cAlias := "SZ9"
```

```
Local cTitulo := "Cadastro de Ordem "
Local cVldExc := ".T."
Local cVldAlt := ".T."

(cAlias)->(dbSetOrder(1))
AxCadastro(cAlias,cTitulo,cVldExc,cVldAlt)

Return Nil
```

Pergunte()

Tela de parâmetros para interação com usuário. A função inicializa as variáveis de pergunta (mv_par01,...) baseada na ordem da pergunta inserida na tabela SX1

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

Argumento	Obrigat.	Tipo	Descrição
cPergunta	Sim	C	Alias do arquivo
IMostra	Não	L	.T. exibirá a tela para edição e carregará as perguntas .F. apenas carregará as perguntas nas variáveis MV_PAR99
cTitle	Não	C	Título da janela de perguntas

Retorno:- Lógico

.T. Clique no botão OK da tela

.F. Clique o botão Cancela da tela

Exemplo:

```
#include "protheus.ch"
User Function ADV0001()

Local cPerg := PADR("AULAXX",10)
If Pergunte(cPerg, .T.)
    Chamafun()
Else
    MsgInfo("Usuario clicou em cancelar", "ADV0001 – Funcao modelo")

Endif

Return Nil
```

PutSx1()

Permite a inclusão de um único item de pergunta em um grupo de definido no Dicionário de Dados (SX1) via código de programação. Todos os vetores contendo os textos explicativos da pergunta devem conter até 40 caracteres por linha.

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

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

Argumento	Obrigat.	Tipo	Descrição
NT	Não	N	Linha Inicial
NI	Não	N	Coluna Inicial
Nb	Não	N	Linha Final
Nr	Não	N	Coluna Final
Calias	Sim	C	Alias do arquivo que será visualizado no browse. Para utilização de arquivos de trabalho, o nome do alias deve ser obrigatoriamente 'TRB' e o parâmetro aFixe torna-se obrigatório.
Afixe	Não	A	<p>Array bi-dimensional contendo os nomes dos campos fixos pré-definidos.</p> <p>A estrutura do array é diferente para arquivos que fazem parte do dicionário de dados e para arquivos de trabalho.</p> <p>Arquivos que fazem parte do dicionários de dados</p> <p>[n][1]=>Descrição do campo [n][2]=>Nome do campo</p> <p>Arquivos de trabalho</p> <p>[n][1]=>Descrição do campo [n][2]=>Nome do campo [n][3]=>Tipo [n][4]=>Tamanho [n][5]=>Decimal [n][6]=>Picture</p>
Ccpo	Não	C	<p>Campo a ser validado se está vazio ou não para exibição do bitmap de status.</p> <p>Quando esse parâmetro é utilizado, a primeira coluna do browse será um bitmap indicando o status do registro, conforme as condições configuradas nos parâmetros cCpo, cFun e aColors.</p>
Nposi	Não	N	
Cfun	Não	C	<p>Função que retornará um valor lógico para exibição do bitmap de status.</p> <p>Quando esse parâmetro é utilizado, o parâmetro cCpo é automaticamente desconsiderado.</p>
Ndefault	Não	N	Número da opção do aRotina que será executada quando for efetuado um duplo clique em um registro do browse. O default é executar a rotina de visualização.

Acolors	Não	A	Array bi-dimensional para possibilitar o uso de diferentes bitmaps de status. [n][1]=>Função que retornará um valor lógico para a exibição do bitmap [n][2]=>Nome do bitmap que será exibido quando a função retornar .T. (True). O nome do bitmap deve ser um resource do repositório e quando esse parâmetro é utilizado os parâmetros cCpo e cFun são automaticamente desconsiderados.
Ctopfun	Não	C	Função que retorna o limite superior do filtro baseado na chave de índice selecionada. Esse parâmetro deve ser utilizado em conjunto com o parâmetro cBotFun .
Cbotfun	Não	C	Função que retorna o limite inferior do filtro baseado na chave de índice selecionada. Esse parâmetro deve ser utilizado em conjunto com o parâmetro cTopFun .
Nfreeze	Não	N	Qtde de colunas que ficarão congeladas
Bparbloco	Não	B	Bloco de código que será executado no ON INIT da janela do browse. O bloco de código receberá como parâmetro o objeto da janela do browse.
Lnotopfilter	Não	L	Valor lógico que define se a opção de filtro será exibida no menu da MBrowse. .T. => Não exibe a opção no menu .F. => (default) Exibe a opção no menu. A opção de filtro na MBrowse está disponível apenas para TopConnect.
Lseeall	Não	L	Identifica se o Browse deverá mostrar todas as filiais. O valor default é .F. (False), não mostra todas as filiais. Caso os parâmetros cTopFun ou cBotFun sejam informados esse parâmetro será configurado automaticamente para .F. (False) Parâmetro válido à partir da versão 8.11. A função SetBrwSeeAll muda o valor default desse parâmetro.
Lchgall	Não	L	Identifica se o registro de outra filial está autorizado para alterações. O valor default é .F. (False), não permite alterar registros de outras filiais. Quando esse parâmetro está configurado para .T. (True), o parâmetro

			<p>ISeeAll é configurado automaticamente para .T. (True).</p> <p>Caso os parâmetros cTopFun ou cBotFun sejam informados esse parâmetro será configurado automaticamente para .F. (False).</p> <p>Parâmetro válido à partir da versão 8.11.</p> <p>A função SetBrwChgAll muda o valor default desse parâmetro.</p>
Cexprfiltop	Não	C	Expressao para efetuar o filtro no browse com sintaxe em TopConnect.

cCadastro- Título que será exibido do browse.

aRotina - Array contendo as funções que serão executadas pela Mbrowse, nele será definido o tipo de operação a ser executada (inclusão, alteração, exclusão, visualização, pesquisa, etc.), e sua estrutura é composta de 5 (cinco) dimensões:

[n][1] - Título;
[n][2] - Rotina;
[n][3] - Reservado;
[n][4] - Operação (1 - pesquisa; 2 - visualização; 3 - inclusão; 4 -alteração; 5 - exclusão);

Exemplo:

```
#include "protheus.ch"
User Function brw00001()

Private cCadastro := "Cadastro de Exemplo"
Private aRotina := {}
AADD(aRotina,{"Pesquisar" ,"AxPesqui",0,1})
AADD(aRotina,{"Visualizar" ,"AxVisual",0,2})
AADD(aRotina,{"Incluir" ,"AxInclui",0,3})
AADD(aRotina,{"Alterar" ,"AxAltera",0,4})
AADD(aRotina,{"Excluir" ,"AxDeleta",0,5})

Private cString := "SA1"

SA1->(dbSetOrder(1))
mBrowse( ,,,cString)

Return
```


Modelo 2

Tela de Manutenção Utilizando uma Tabela (Cabecalho/Itens).

Sintaxe:

Modelo2(cTitulo,aCab,aRod,aGetdados,[nOpc],[cLinOk],[cTudOk],aCols,[bF4],[cInitPos],[nMax],[aCoord],[lDelGet],[lMaximized],[aButtons])

Argumento	Obrigat.	Tipo	Descrição
cTitulo	Sim	C	Título da Janela
aCab	Sim	A	Array com dados do cabeçalho Ex: {cVariavel,{nLin,nCol},cTitulo_Campo,cPicture,Funcao_valid,F3,lEditavel}
aRod	Sim	A	Array com dados do rodapé Ex: {cVariavel,{nLin,nCol},cTitulo_Campo,cPicture,Funcao_valid,F3,lEditavel}
aGetdados	Sim	A	Array com posições da getdados Ex: {nLinIni,nColIni,nLinFim,nColFim}
nOpc	Nao	N	Modo de operação (3/4 Inclui ou altera, 6 altera, outros visualiza) Ex: nOpc := 3
cLinOk	Não	C	Função para validar linha Ex: "allwaystrue()"
cTudOk	Nao	C	Função que valida tudo Ex: "allwaystrue()"
aCols	Sim	A	Array com os campos que podem ser alterados Ex: {"A1_COD","A1_NREDUZ"}
bF4	Não	B	Bloco de Código Chamado pelo F4 Ex: { MsgAlert("Teste")}
cInitPos	Não	C	String com nome dos campos que devem ser inicializados ao teclar seta para baixo.
nMax	Não	N	Número máximo de linhas
aCoord	Não	A	Array com as coordenadas das janelas
lDelGet	Não	L	Determina se a linha da getdados pode ser apagada ou não
lMaximized	Não	L	Maximiza a tela
aButtons	Não	A	Array com botões a serem acrescentados no toolbar

Retorno:- Lógico

.T. Clique no botão OK da tela

.F. Clique o botão Cancela da tela

Modelo 3

Tela de Manutenção Utilizando duas Tabelas (Cabecalho/Itens).

Argumento	Obrigat.	Tipo	Descrição
cTitulo	Sim	C	Título da Janela
cAlias1	Sim	C	Alias da tabela do Cabeçalho
cAlias2	Sim	C	Alias da Tabela dos Itens
aEnchoice	Sim	A	Array com os campos da enchoice(Cabecalho)
cLinOk	Não	C	Função de Validação de Itens
cTudOK	Não	C	Função de Validação tudo
nOpc1	Não	N	Opção de edição da Enchoice
nOpc2	Não	N	Opção de edição da Getdados
cFieldOk	Não	C	Função de validação dos campos da Getdados
lVirtual	Não	L	Permite visualizar campos virtuais na Enchoice
nLinhas	Não	N	Numero Maximo de Linhas da Getdados
aAltEnchoice	Não	A	Array com os campos alteráveis da enchoice
nFreeze	Não	N	Qtde de colunas que ficarão congeladas
aButtons	Não	A	Array com botões a serem acrescentados ao toolbar
aCoord	Não	A	Array com as coordenadas da tela
nSize	Não	N	Numero de linhas da enchoice

Retorno:- Lógico

.T. Clique no botão OK da tela

.F. Clique o botão Cancela da tela

Exemplo:

```
#include "protheus.ch"
User Function CadM3()
Private cAlias1 := "SC5"
Private cAlias2 := "SC6"
Private aHeader := {}
Private aCols := {}
Private cCadastro:= "Modelo 3"
Private aRotina := {}
AAdd(aRotina, {"Pesquisar", "axPesqui" , 0, 1})
AAdd(aRotina, {"Visualizar", "axVisual" , 0, 2})
AAdd(aRotina, {"Incluir" , "u_Incmod3" , 0, 3})
AAdd(aRotina, {"Alterar" , "u_Incmod3" , 0, 4})
AAdd(aRotina, {"Excluir" , "u_Incmod3" , 0, 5})

mBrowse(,,,cAlias1)

Return

User Function Incmod3(cAlias, nReg, nOpc)

//Criar as variaveis de memoria
RegToMemory(cAlias1, (nOpc==3))

//Monta aheader
SX3->(dbSetorder(1))
SX3->(dbSeek(cAlias2 ))

While SX3-> (!EOF() .AND. SX3->X3_ARQUIVO == cAlias2)

    If X3USO(SX3->X3_USADO) .AND. cNIVEL >= SX3->X3_NIVEL

        AAdd (aHeader, {Trim(SX3->X3_TITULO),; //01 - titulo
            SX3->X3_CAMPO ,; //02- nome do campo
            SX3->X3_PICTURE ,; //03 - mascara do campo
            SX3->X3_TAMANHO ,; //04 - tamanho
            SX3->X3_DECIMAL ,; //05 - decimais
            SX3->X3_VALID ,; //06 - validacao
            SX3->X3_USADO ,; //07 - USADO
            SX3->X3_TIPO ,; //08 - TIPO DO CAMPO
            SX3->X3_ARQUIVO ,; //09 - ALIAS
            SX3->X3_CONTEXT}) //10 - Virtual ou Real

    ENDIF
```

```
SX3->(DBSKIP())

ENDDO

//Monta acols
If nOpc == 3 //Inclusão
    AAdd(aCols, Array(Len(aHeader) + 1))
    For nAdi:= 1 to Len(aHeader)
        aCols[1][nADI] := Criavar(aHeader[nADI][2])
    Next
    ACols[1][len(aHeader)+1] := .F.

Else //Visual/Altera/Exclui
    (cAlias2)->(dbSetOrder(1))
    (cAlias2)->(dbSeek(xFilial(cAlias2) + (cAlias1)->C5_NUM))
    While (cAlias2)->(EOF()).And.(cAlias2)->C6_FILIAL == xFilial(cAlias2)
.And.;
        (cAlias2)->C6_NUM== (cAlias1)->C5_NUM

        AAdd(aCols, Array(len(aHeader)+1))

        For i := 1 To nQtdCpo
            If aHeader[i][10] <> "V"
                aCols[len(aCols)][i] := (cAlias2)->&(aHeader[i][2])
            Else
                aCols[len(aCols)][i] := Criavar(aHeader[i][2])
            EndIf
        Next
        ACols[1][len(aHeader)+1] := .F.
    Endif

    IRet := Modelo3(cCadastro, cAlias1, cAlias2, "Allwaystrue", "Allwaystrue" ,
nOpc, nOpc)

Return
```

Exercício

Fazer um programa que trate a Amarração Cliente x Produto

Tabelas Envolvidas

- Cadastro de Clientes (SA1) , sempre visual (não permite alteração)
- Amarração Produtos x Clientes (SA7) Obedece a escolha do usuário no browse

Browse (Cadastro de Clientes)

Itens do aRotina

- Incluir, Alterar ,visualizar , excluir os dados
- Pesquisa Padrão (AXPESQUI)
- Não permitir incluir o mesmo produto para o mesmo cliente.
- Não permitir alteração caso não exista nenhum item amarrado para o cliente selecionado.
- Na exclusão, excluir apenas os itens (SA7), Não deixar excluir caso não exista nenhum item amarrado para o cliente selecionado.
- Trazer o acols já preenchido conforme registro posicionado pelo usuário.

Objetivo

Capítulo 18 – MSEXECAUTO

Fazer manutenção automática (inclusão, alteração e exclusão) das rotinas de manipulação de dados do sistema, automatizando o processo de entrada de dados sem a necessidade de desenvolver rotinas específicas.

Vantagens

- 1) Interface : Os dados de entrada são enviados a rotina em forma de campos e conteúdos (array) e desta forma não é necessário a apresentação de nenhuma interface ao usuário.
- 2) Segurança : A utilização de rotinas automáticas aumenta consideravelmente a segurança do sistema, uma vez que utiliza as validações padrões e diminui os problemas causados por atualização de versão ou inclusão de customizações nas rotinas padrões do sistema.
- 3) Agilidade no processo : Aumenta consideravelmente o tempo de desenvolvimento das customizações que necessitam de entrada de dados. Exemplo: Importação de pedido de venda.

Procedimentos

Sintaxe: MSEXecAuto({|x,y|fpgmpad(x,y)},aCampos,nOpc)

Argumento	Obrigat.	Tipo	Descrição
fPgmPad	Sim	B	Nome da Função Padrão
aCampos	Sim	A	Array com os campos a gravar
nOpc	Não	N	Opção do Browse que será utilizada.

Existem duas maneiras de utilizar a rotina automática, sendo elas:

➤ Sem Interface

Para a utilização da rotina automática sem interface deve-se, configurar o ambiente utilizando-se o comando PREPARE ENVIRONMENT e chamar diretamente o nome da função.

Exemplo:

```
#include "protheus.ch"
User Function IncProd()
Local aRotAuto := {}
Local nOpc := 3 // inclusao
Private IMsHelpAuto := .t. // se .t. direciona as mensagens de help para o arq. de log
Private IMsErroAuto := .f. //necessario a criacao, pois sera atualizado quando houver
//alguma inconsistencia nos parametros

PREPARE ENVIRONMENT EMPRESA '99' FILIAL '01' MODULO 'FAT'
aRotAuto:= {{'B1_COD' ,GETSXENUM('SB1', 'B1_COD') ,Nil},,
{'B1_DESC' , 'Produto teste',Nil},,
{'B1_TIPO' , 'PA' ,Nil},,
{'B1_UM' , 'UN' ,Nil},,
```

```

        {'B1_LOCPAD' , '01' , Nil},,
        {'B1_PICM' , 0 , Nil},,
        {'B1_IPI' , 0 , Nil},,
        {'B1_PRV1' , 100 , Nil},,
        {'B1_LOCALIZ' , 'N' , Nil},,
        {'B1_CODBAR' , '789888800001' , Nil}}
MSEExecAuto({|x,y| mata010(x,y)},aProduto,nOpc)
If IMsErroAuto
    //mostrar na tela o log informando qual a inconsistência ocorreu durante a
    rotina
    Mostraerro()
    Return .f.
EndIf
Return .t.

```

➤ Com Interface

Para a utilização da rotina automática com interface deve-se apenas colocar no menu.

Exemplo:

```

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

Local aRotAuto := {}
Local nOpc := 3 // inclusao
Private IMsHelpAuto := .t. // se .t. direciona as mensagens de help para o arq. de log
Private IMsErroAuto := .f. //necessario a criacao, pois sera atualizado quando houver
    //alguma incosistencia nos parametros

aRotAuto:= {'B1_COD' ,GETSXENUM('SB1' , 'B1_COD') , Nil},,
           {'B1_DESC' , 'Produto teste', Nil},,
           {'B1_TIPO' , 'PA' , Nil},,
           {'B1_UM' , 'UN' , Nil},,
           {'B1_LOCPAD' , '01' , Nil},,
           {'B1_PICM' , 0 , Nil},,
           {'B1_IPI' , 0 , Nil},,
           {'B1_PRV1' , 100 , Nil},,
           {'B1_LOCALIZ' , 'N' , Nil},,
           {'B1_CODBAR' , '789888800001' , Nil}}
MSEExecAuto({|x,y| mata010(x,y)},aProduto,nOpc)

If IMsErroAuto
    //mostrar na tela o log informando qual a inconsistência ocorreu durante a
    rotina

```

```
Mostraerro()  
Return .f.  
  
EndIf  
Return .t.
```

Onde usar:

Existem varias situações em que podemos usar:

- Customizações de rotinas que automatizam processos entre empresas e facilitam o trabalho do usuário.
- Importação de dados para o sistema.
 - Arquivo texto,
 - XML,
 - Sites da Web,
 - ADVPL ASP,
 - Web Services.
 - Workflow
- Replicação de Dados, como cópia de pedidos, produtos, clientes, fornecedores, liberação de pedidos, etc

Exemplo de Automação de Rotinas**Tipo de Entrada e Saída:**

```
#include "protheus.ch"  
User Function IncTes()  
  
Local aRotAuto := {}  
Local nOpc := 3 // inclusao  
Private IMsHelpAuto := .t.  
Private IMsErroAuto := .f.  
  
//Entrada  
aTes:={{"F4_CODIGO"    ,"011",Nil},;  
       {"F4_TIPO"      ,"E",Nil},;  
       {"F4_ICM"        ,"S",Nil},;  
       {"F4_IPI"        ,"S",Nil},;  
       {"F4_CREDICM"    ,"S",Nil},;  
       {"F4_CREDIPI"    ,"S",Nil},;  
       {"F4_DUPLIC"     ,"S",Nil},;  
       {"F4_ESTOQUE"    ,"S",Nil},;  
       {"F4_CF"         ,"1102",Nil},;  
       {"F4_TEXTO"      ,"COMPRA",Nil},;
```



```

{"F4_PODER3"      ,"N",Nil},;
{"F4_LFICM"      ,"T",Nil},;
{"F4_LFIPI"      ,"T",Nil},;
{"F4_DESTACA"    ,"N",Nil},;
{"F4_INCIDE"     ,"N",Nil},;
{"F4_COMPL"     ,"N",Nil}}
MSExecAuto({|x,y| mata080(x,y)},aTes,nOpc)

//Saída
aTes:={{ "F4_CODIGO"      ,"511",Nil},;
{"F4_TIPO"      ,"S",Nil},;
{"F4_ICM"      ,"S",Nil},;
{"F4_IPI"      ,"S",Nil},;
{"F4_CREDICM"   ,"S",Nil},;
{"F4_CREDIPI"   ,"S",Nil},;
{"F4_DUPLIC"    ,"S",Nil},;
{"F4_ESTOQUE"   ,"S",Nil},;
{"F4_CF"        ,"5102",Nil},;
{"F4_TEXTO"     ,"COMPRA",Nil},;
{"F4_PODER3"    ,"N",Nil},;
{"F4_LFICM"     ,"T",Nil},;
{"F4_LFIPI"     ,"T",Nil},;
{"F4_DESTACA"   ,"N",Nil},;
{"F4_INCIDE"    ,"N",Nil},;
{"F4_COMPL"     ,"N",Nil}}
MSExecAuto({|x,y| mata080(x,y)},aTes,nOpc)

If IMsErroAuto
  Mostraerro()
  Return .f.
EndIf
Return .t.

```

Documento de Entrada

```

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

Local aRotAuto := {}
Local nOpc := 3 // inclusao
Private IMsHelpAuto := .t.
Private IMsErroAuto := .f.
aCab := {{ "F1_TIPO"      ,'N'      ,NIL},;

```

```

{"F1_FORMUL"      ,"N"              ,NIL},;
{"F1_DOC"         ,"000001"         ,NIL},;
{"F1_SERIE"       ,"UNI"            ,NIL},;
{"F1_EMISSAO"     ,dDataBase         ,NIL},;

{"F1_FORNECE"     ,"000001"         ,NIL},;
{"F1_LOJA"        ,"01"             ,NIL},;
{"F1_COND"        ,"001"            ,NIL},;
{"F1_ESPECIE"     ,"NF"             ,NIL}}

```

```

For nx:=1 to 3
  aLinha:={}
  AADD(aLinha,{"D1_COD"      ,"EX"+Str(nx,1,0),NIL})
  AADD(aLinha,{"D1_UM"       ,"PC"              ,NIL})
  AADD(aLinha,{"D1_QUANT",10000              ,NIL})
  AADD(aLinha,{"D1_VUNIT",1                  ,NIL})
  AADD(aLinha,{"D1_TOTAL",10000              ,NIL})
  AADD(aLinha,{"D1_TES"      ,"001"           ,NIL})
  AADD(aLinha,{"D1_LOCAL","01"              ,NIL})
  AADD(aItens,aLinha)
Next nx
MSEExecAuto({|x,y,z| MATA103(x,y,z)},aCab,aItens,nOpc)
If IMsErroAuto
  Mostraerro()
  Return .f.
EndIf
Return .t.

```

Fornecedores

```

#include "protheus.ch"

User Function IncFor()
Local aRotAuto := {}
Local nOpc := 3 // inclusao
Private IMsHelpAuto := .t.
Private IMsErroAuto := .f.

aFornecedor:={      {"A2_COD"      ,GETSXENUM('SA2',
'A2_COD'),Nil},;
                    {"A2_LOJA"    ,"01",Nil},;
                    {"A2_NOME"    ,"FORNECEDOR VENDE BARATO

```

S.A.",Nil},,;

{"A2_NREDUZ","VENDE TUDO",Nil},,;

{"A2_TIPO" ,"F",Nil},,;

{"A2_END" ,"AV LEONARDO DA VINCE, 608",Nil},,;

{"A2_MUN" ,"SAO PAULO",Nil},,;

{"A2_EST" ,"SP",Nil},,;

{"A2_COND" ,"001",Nil},}

MSExecAuto({|x,y| mata020(x,y)},aFornecedor,nOpc)

If IMsErroAuto

Mostraerro()

Return .f.

EndIf

Return .t.

Cientes

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

```
User Function IncCli()
```

```
Local aRotAuto := {}
```

```
Local nOpc := 3 // inclusao
```

```
Private IMsHelpAuto := .t.
```

```
Private IMsErroAuto := .f.
```

```
    aCliente:= { {"A1_COD"    ,GETSXENUM('SA1', 'A1_COD'),Nil},;  
                {"A1_LOJA"   ,"01",Nil},;  
                {"A1_NOME"    ,"CLIENTE COMPRA TUDO S.A.",Nil},;  
                {"A1_NREDUZ", "COMPRA TUDO",Nil},;  
                {"A1_TIPO"    ,"F",Nil},;  
                {"A1_END"     ,"RUA DO ROCIO 123",Nil},;  
                {"A1_MUN"     ,"SAO PAULO",Nil},;  
                {"A1_EST"     ,"SP",Nil},;  
                {"A1_COND"    ,"001",Nil}}
```

```
    MSExecAuto({|x,y| mata030(x,y)},aCliente,nOpc)
```

```
    If IMsErroAuto
```

```
        Mostraerro()
```

```
        Return .f.
```

```
    EndIf
```

```
Return .t.
```

Exercícios

1. Criar uma rotina que inclua automaticamente pelo MSEXCAUTO um pedido de vendas com 2 itens quaisquer.

Apoio: ROT_AUTO.PRW

Funcao de inclusao:mata410()

Tabelas: SC5 e SC6

2. Criar uma rotina automática para o cadastro de Clientes, para isso crie um grupo de perguntas com as seguintes perguntas:

- Nome, Endereco, CNPJ, telefone, tipo
- Use o MSEXecauto com esses campos.
- Coloque no aRotina do Browse do exercício anterior.
- Lembre-se de gravar o código usando a função GETSXENUM()
- Mostre na tela o código gerado