TOTVS

DANILO FERNANDO DIAS– Desafio da Capacitação

Melhores Práticas de Programação em AdvPL

TOTVS

Índice

Introdução

TOTVS

Introdução

Programar utilizando regras de boas práticas é fundamental para garantir qualidade e facilitar a manutenção e entendimento dos fontes que desenvolvemos. Um fonte bem escrito e estruturado pode reduzir erros e o tempo para efetuar manutenções e melhorias.

Todo analista deve se preocupar em utilizar essas regras nos códigos que escreve, tendo isso como algo natural.

Esse treinamento tem como objetivo definir convenções, padrões, práticas e técnicas que melhorem a qualidade e a manutenção dos softwares criados utilizando a linguagem ADVPL e o Microsiga Protheus.

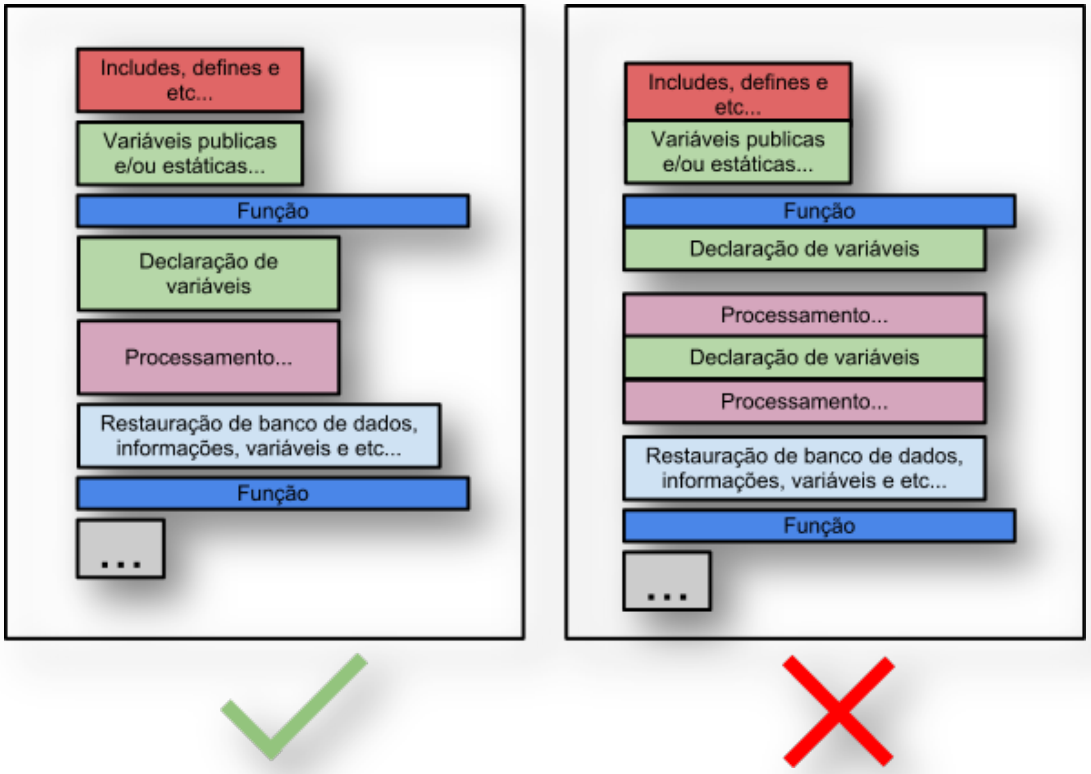
Conteúdo

TOTVS

Formatação do Código Fonte

Formatação Vertical

A formatação vertical representa a distância entre as declarações de importação, variáveis, funções, e assim por diante. Nós conseguimos uma leitura mais rápida e clara quando enxergamos pequenos blocos e cada bloco com o conteúdo de mesmo tipo.

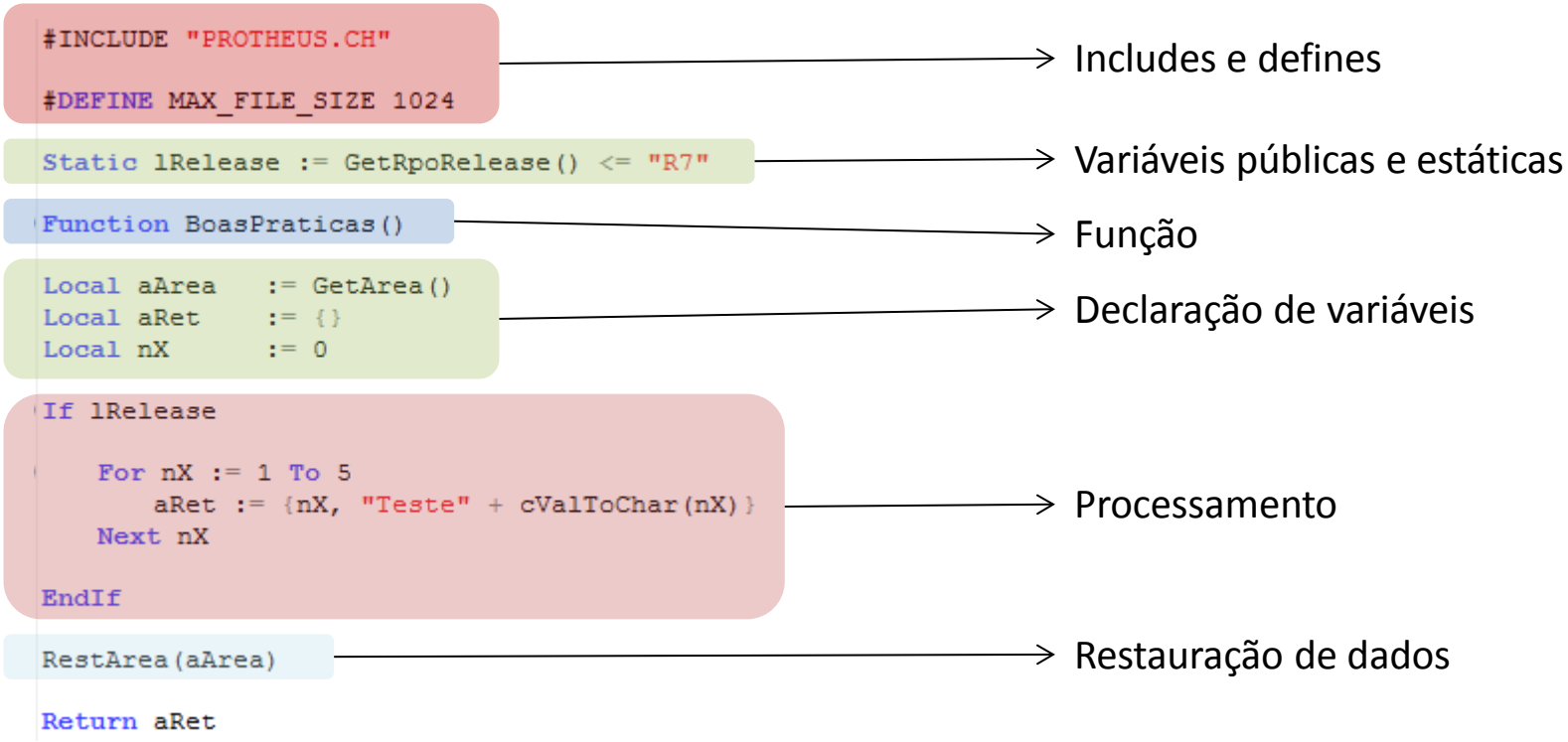


TOTVS

Formatação do Código Fonte

Formatação Horizontal

A formatação horizontal convenciona o espaçamento, posições e alinhamento horizontal entre os comandos e expressões da linguagem.



Capitulação Heterogênea

Usada para facilitar a leitura em nomes compostos nos códigos fontes, mesclando letras maiúsculas e minúsculas. A capitulação heterogênea deve ser usada para os seguintes casos:

- Nomes de variáveis (Exceto a sigla da notação húngara);
- Funções;
- Classes;
- Métodos;
- Comandos;
- Siglas.

TOTVS

Capitulação de Palavras Chave

Exemplos de código fonte com e sem capitulação.

```
Function CapitulacaoHeterogenea()

Local dDataNascimento    := ""
Local nIdadePessoa       := 0
Local cNomePessoa        := ""
Local aPessoas           := {}

cNomePessoa       := PerguntaNome()
dDataNascimento   := PerguntaDataNascimento()
nIdadePessoa      := CalculaIdade( cDataNascimento )

If ( !Empty( cNomePessoa ) .And. nIdadePessoa > 0 )
   AAdd( cNomePessoa, nIdadePessoa )
EndIf

Return aPessoas
```

```
function semcapitulacaoheterogenea()

local ddatanascimento    := ""
local nidadepessoa       := 0
local cnomepessoa        := ""
local apessoas           := {}

cnomepessoa       := perguntanome()
ddatanascimento   := perguntadatanascimento()
nidadepessoa      := calculaidade( cdatanascimento )

if ( !empty( cnomepessoa ) .and. nidadepessoa > 0 )
   aadd( cnomepessoa, nidadepessoa )
endif

return apessoas
```

TOTVS

Capitulação de Palavras Chave

- AAdd, ADel e ASize estão relacionadas com a manipulação de array, o “A” é apenas a abreviação de “Array”, devendo ser escrito em maiúsculo.

- DbSeek, DbGoTo, DbCreate estão relacionadas com a manipulação de banco de dados, o composto “banco de dados” em inglês é escrito junto “Database”, portanto Db é a abreviação de database e não uma sigla ou as iniciais de duas palavras, não devendo ser escrita como “DBSeek”.

Capitulação Homogênea

A capitulação homogênea deve ser usada sempre maiúscula nos seguintes casos:

- Constantes;
- Variáveis de memória;
- Campos;
- Queries;
- Comandos de pré-processamento.

TOTVS

Capitulação de Palavras Chave

Exemplo de código fonte com capitulação homogênea.

```
#INCLUDE "PROTHEUS.CH"

#DEFINE INCLUSAO  3
#DEFINE ALTERACAO 4
#DEFINE EXCLUSAO  5

Function CapitulacaoHomogenea( nOpc )

Local cCodigo := M->A1_COD
Local cQuery  := ""

If ( nOpc == INCLUSAO )

    cQuery := "SELECT SA1_NOME FROM " + RetSQLName("SA1")
    cQuery += " WHERE SA1_COD = '" + cCodigo + "'"
    cQuery += " AND SA1_LOJA = '" + M->A1_LOJA + "'"

EndIf

Return
```

Comentários

Comentários servem como documentação e explicação de um código fonte, facilitando seu entendimento para análise e manutenção.

Comentar o fonte, facilita o entendimento por parte do analista que precisar realizar manutenção e até mesmo para quem o criou, mas lembre-se que comentário em excesso polui o código e pode atrapalhar, portanto use o bom senso.

Não existe uma regra sobre o que se deve comentar ou como, mas segue algumas dicas sobre o que comentar e melhorar o entendimento do seu código fonte.

Discussões sobre decisões não obvias ou não triviais no design do código-fonte são bem vindas, porém evite duplicar informações que já estão presentes de forma clara no código-fonte. Comentários redundantes podem facilmente se tornarem desatualizados, evite comentar informações que tem o potencial de se tornarem desatualizados

TOTVS

Comentários

- Insira um cabeçalho e comente qual finalidade da rotina ou função, descrevendo quais parâmetros devem ser passados e qual deve ser o retorno esperado, utilize o padrão ProtheusDoc.
- Comente a chave do índice utilizado ao lado da função DbSetOrder()
- Quando utilizar strings de um arquivo CH, insira um comentário ao lado com o texto da string.
- Comente cálculos, explicando qual sua lógica.
- Comente trechos de código, explicando de forma sucinta sua finalidade.
- Não comente código fonte removido, ao invés disso apague a linha, o TFS mantém histórico de versões para consultar alterações feitas.

TOTVS

Comentários

Comentários simples "//" devem estar localizados no final da linha comentada ou no começo de um bloco de código, indicando assim, a intenção e objetivos.

```
Function ComentariosSimples()

//Exemplo de Loop
While .T.

    If .T.
        ConOut( "Verdadeiro" )   //Faz alguma coisa
    Else
        Exit   //Termina
    EndIf

EndDo

Return
```

TOTVS

Comentários

Comentários de múltiplas linhas

Comentários de múltiplas linhas "/* */" não devem ser utilizados (com exceção do ProtheusDoc), ao invés, aplicar sequencialmente comentários simples e delimitar o inicio e o fim do comentário com uma sequencia sem tamanho definido de "-".

```
//-------------------------------------------------------------------
/*/{Protheus.doc} ComentariosMultiplasLinhas
Demonstra o uso de comentários em múltiplas linhas.

@sample    ComentariosMultiplasLinhas( nCat1, nCat2 )
@param     nCat1 - Cateto 1
           nCat2 - Cateto 2
@return    nHipo - Hipotenusa

@author    TOTVS
@since     12/06/2012
@version   P12
/*/
//-------------------------------------------------------------------
Function ComentariosMultiplasLinhas( nCat1, nCat2 )

//-------------------------------------
// Calcula a Hipotenusa de um triângulo
// retângulo de acordo com o teorema de
// Pitágoras onde, H² = C1² + C2².
//-------------------------------------
nHipo = SqRt( ( nCat1 ^ 2 ) + ( nCat2 ^ 2 ) )

Return nHipo
```

Declaração de Variáveis

Notação Húngara

A Notação Húngara visa facilitar o reconhecimento do tipo da variável em um código-fonte. A adoção deste critério de nomeação é bastante prática e intuitiva, sendo a ideia básica nomear todos os tipos de variáveis, visando-se simplificar o entendimento. Para isso, deve-se adicionar um prefixo ao nome da variável. Esse prefixo será em letra minúscula e de acordo com o tipo de conteúdo da mesma.

Notação	Tipo de Dado	Exemplo
a	Array	aParcelas
c	Caracter	cNome
d	Data	dDataFim
l	Lógico	lGravou
n	Numérico	nValor
o	Objeto	oDialog
u	Indefinido	uBuffer

Declaração de Variáveis

As variáveis nunca devem ser declaradas na mesma linha, independente da quantidade ou de serem todas do mesmo tipo. Isso garante uma melhor visualização e um espaço para comentários.

A posição do bloco deve ser no inicio da função.

Todas as variáveis devem ser inicializadas na sua declaração. Por padrão os seguintes tipos de variáveis devem ser inicializados com os seguintes conteúdos:

```
Local aArray   := {}                    //Array
Local bBloco   := {||}                  //Bloco de Código
Local cTexto   := ''                    //Caracter
Local dData    := DToC('  /  /  ')      //Data
Local lLogico  := .F. ou .T.            //Lógico
Local nNumero  := 0                     //Numérico
Local oObjeto  := Nil                   //Objeto
Local uGeral   := Nil                   //Indefinido
```

TOTVS

Declaração de Variáveis

Formatação de Tipo

Arrays

Arrays podem guardar tanto estruturas simples quando estruturas complexas e em ambos os casos se não houver uma formatação clara e legível, a sua programação e manutenção podem se tornar difíceis e geradoras de problemas. No momento de atribuição de valores em arrays, recomendamos o uso da formatação horizontal para ficar claro para o leitor como é a estrutura de dimensões da array:

```
aArray := {{ "1", 0, Date() },;
           { "2", 1, Date() },;
           { "3", 2, Date() }}
```

TOTVS

Declaração de Variáveis

Quando o array é utilizado para representar um mapeamento de chave e valor, e seu tamanho for definido, sugerimos a utilização de defines para nomear as posições da array:

```
#Include 'Protheus.ch'

#DEFINE PARCELA 1
#DEFINE ID 2
#DEFINE VENCIMENTO 3

Function ExemploArray()

Local aArray := {}

aArray := {{ "1", 0, Date() },;
           { "2", 1, Date() },;
           { "3", 2, Date() }}

ConOut( "Parcela: " + aArray[PARCELA] )
ConOut( "Id: " + CValToChar(aArray[ID]) )
ConOut( "Vencimento: " + DToC(aArray[VENCIMENTO]) )

Return
```

TOTVS

Declaração de Funções

Nas funções, tanto a declaração quando a chamada deve ser feita utilizando a capitulação heterogênea. Não deve haver espaçamento entre o parêntese e o parâmetro, e deve haver um espaço adicional somente após a vírgula que divide os parâmetros.

Quando houver mais do que 8 parâmetros recebidos. Deve-se separa-los 8 em 8, portanto, 8 na linha de declaração da função, 8 na outra linha alinhados com os anteriores e assim por diante.

```
Function ExemploFuncao(cParam1, cParam2, cParam3, cParam4, cParam5, cParam6, cParam7, cParam8,;
                       cParam9, cParam10, cParam11, cParam12)

FuncaoTeste(cParam1, cParam2)

Return
```

TOTVS

Queries

Construção Simples

Quando a construção de uma query for simples, trate a formatação horizontal do mesmo modo como se é tratada na construção de funções e classes. Por exemplo:

```
cQuery := "SELECT A1_COD, A1_NOME"
cQuery += "  FROM " + RetSQLName("SA1")
cQuery += " WHERE A1_FILIAL = '" + xFilial("SA1") + "'"
cQuery += "   AND D_E_L_E_T_ = ' '"
```

TOTVS

Queries

Construção Condicional

Na construção condicional de queries é importante manter agrupado em um bloco sequencial. Não permita que processamentos ou comandos não relativos a montagem da query esteja intercalado ao bloco de montagem de query.

```
cQuery := "SELECT A1_COD, A1_NOME"
cQuery += "  FROM " + RetSQLName("SA1")
cQuery += " WHERE A1_FILIAL = '" + xFilial("SA1") + "'"

If (lOnlyNotDeleted)
    cQuery += "   AND D_E_L_E_T_ = ' '"
EndIf
```

TOTVS

Embedded SQL

A formação aplicada no *embedded* SQL segue a formatação horizontal padrão e por ser um comando, deve seguir a capitulação heterogênea:

```
BeginSQL Alias cAliasTrb

       SELECT R_E_C_N_O_ RECNOSN1
         FROM %Table:SN1%
        WHERE N1_FILIAL = %XFilial:SN1%
          AND N1_CBASE >= %Exp:MV_PAR01%
          AND N1_CBASE <= %Exp:MV_PAR02%
          AND N1_ITEM  >= %Exp:MV_PAR03%
          AND N1_ITEM  <= %Exp:MV_PAR04%
          AND N1_GRUPO >= %Exp:MV_PAR05%
          AND N1_GRUPO <= %Exp:MV_PAR06%
          AND N1_AQUISIC >= %Exp:MV_PAR07%
          AND N1_AQUISIC <= %Exp:MV_PAR08%
          AND %Exp:cWhere%
          AND %NotDel%

EndSQL

Return
```

TOTVS

Nomes Longos

A linguagem ADVPL suporta para nomes de funções, métodos, variáveis, comandos de pré-processamento o tamanho máximo de 10 caracteres. A única exceção é nome de classe que não tem limite.

Para minimizar o impacto dessa limitação, não recomendamos que o nome seja abreviado, recomendamos a inversão do nome, para que a palavra que diferencia duas funções esteja no começo do nome, por exemplo:

Ao invés de:

```
OpNTaxRece()          // Open No Tax Receipt (Abre cupom não fiscal);
ClBoNTaxRe()          // Close Bound No Tax Receipt (Fecha cupom não fiscal vinculado).
```

Utilizar:

```
OpenNoTaxReceipt()    // Open No Tax Receipt (Abre cupom não fiscal);
CloseBoundNoTaxReceipt()  // Close Bound No Tax Receipt (Fecha cupom não fiscal vinculado).
```

Ao invés de:

```
OpenNoTaxReceipt()    // Abre cupom não fiscal;
OpenNoTaxReceiptBound()  // Abre cupom não fiscal vinculado.
```

Utilizar:

```
OpenNoTaxReceipt()    // Abre cupom não fiscal;
OpenBoundNoTaxReceipt()  // Abre cupom não fiscal vinculado.
```

TOTVS

Padrões de Design

Nomeação dos Módulos

Por convenção, o nome do módulo não excede oito caracteres.

SIGAXXX

SIGA - Obrigatório
XXX - Sufixo do nome real do módulo.

Nomeação de Projetos

Para nomes de Projetos, usamos uma sintaxe para padronizar, conforme explicado a seguir.

SIGWXXX.PRJ

SIGW - É fixo do sistema e obrigatório.
XXX - Sufixo do nome real do módulo.

Nomeação de Códigos Fonte

Os programas da linha Microsiga Protheus devem possuir 7 (sete) dígitos e duas extensões possíveis, conforme a explicado a seguir:

XXXYNNN[III][S].PRW

XXX - Prefixo do módulo da linha Microsiga Protheus, exemplo: GPE para o módulo de Gestão de Pessoal, PON para o módulo de Ponto Eletrônico, FAT para o módulo de Faturamento, etc.

Y - Código que identifica a operação do programa. Sendo:

A	Formulários ou Processamento
C	Consulta de dados
R	Relatórios
X	Bibliotecas
M	Miscelâneas

TOTVS

Padrões de Design

NNN - Código sequencial do programa. Mantenha o padrão de numeração de 10 em 10 e relacione as operações de programa similares. Exemplos:

GPEA010 – Cadastros de funcionários;
GPEA020 – Cadastro de pessoas;
GPER010 – Relatório de funcionários;
GPER020 – Relatório de pessoas;

III - As três últimas letras identificam a localização de origem do programa. Devem ser utilizados com base na tabela da norma ISO 3166 (http://tdn.totvs.com/pages/viewpage.action?pageId=22480523).

Atenção: A sigla torna-se obrigatória apenas para programas de produtos localizados. Caso contrário a sigla deixa de ser obrigatória.

S - Sequenciador da rotina. Algumas rotinas complexas necessitam mais do que somente um código-fonte, nesses casos deve ser utilizado um sequenciador alfanumérico. Por exemplo, a rotina de televendas utiliza quatro programas, sendo:

TMKA273A.PRW
TMKA273B.PRW
TMKA273C.PRW

PRW - Extensão PRW. Arquivos de rotinas antigas ainda mantém o uso da extensão prx. Em mudanças de versão do Microsiga Protheus essas rotinas devem ser renomeadas para a extensão prw.

Nomeação de Funções

Para funções auxiliares de uma rotina, relatório ou consulta, utilizar o padrão:

XYNNNXXXXX()

X - Primeira letra do nome do código-fonte.

Y - Tipo de Operação indicado no código-fonte.

NNN - Identificador do código-fonte.

XXXXX - Abreviação descritiva do objetivo da função.

Exemplo: Função de inclusão de dados presente FINA050: FA050Inclu().

TOTVS

Padrões de Design

Nomeação de Funções

Para funções genéricas de utilização pelo módulo, utilizar o padrão:

XXZZZZZZZZ()

XX - Prefixo do módulo da linha Microsiga Protheus, exemplo: GP para o módulo de Gestão de Pessoal, PO para o módulo de Ponto Eletrônico, FT para o módulo de Faturamento, etc;

ZZZZZZZZ - Abreviação descritiva do objetivo da função.

Exemplo: Função genérica do módulo Faturamento: FTFuncaoGenerica().

Nomeação de Funções

Funções genéricas de utilização pela equipe do produto, utilizar o padrão:

XXZZZZZZZZ()

XX - Abreviação no nome da equipe, exemplo: VC para Vendas e CRM, MT para Materiais, etc;
ZZZZZZZZ - Abreviação descritiva do objetivo da função.

Exemplo: Função genérica da equipe de Recursos Humanos: RHFuncaoGenerica().

TOTVS

Padrões de Design

Nomeação de Classes e Métodos

As classes devem ser iniciadas com abreviação ou sufixo do módulo, Pacote que a classe pertence (opcional), Tipo de Classe ()e o nome da classe, sem limite de tamanho, então recomendamos que o nome seja o máximo sugestivo possível e sem abreviações. Veja:

“FT” – Abreviação - Faturamento;

“BC” - Pacote - *Business Component* , que é um pacote que podemos identificar que a classe tem a função de Regra de Negócios;

“A” - Tipo de Classe, “A” para Abstrata e “C” para Concreta.

“GeraFaturantoSimples” - Nome da classe

Ficamos então: FTBCAGeraFaturamento

Os nomes do pacote que pertence uma Classe não são obrigatórios, porem é recomendável para que numa consulta saibamos rapidamente qual a função básica dessa classe, então segue alguns exemplos:

“BC” - *Business Component* - Classe com funcionalidade voltada para Regra de Negócio

“DT” - *Data* - Classe com funcionalidade voltada para dados, Consulta, Alteração e Inclusão

“BW” - *Busines Work Flow* - Classe que reponsável pelo Fluxo de chamadas dos componentes.

Importante também, que para melhor organização e localização de uma Classe, temos que ter sempre um arquivo (fonte) por classe.

Os Métodos, pode ser composto somente pelo nome, respeitando tamanho de 10 caracteres.

Nomeação de Parâmetros

Os parâmetros criados por um módulo devem seguir a seguinte regra de identificação:

MV_YYZZZZZ

MV - Indica que é um parâmetro do módulo.
YY - Indica de qual módulo é o parâmetro.
ZZZZZ - Nome dado ao parâmetro.

Exemplos:

Módulo	Parâmetro
SIGALOJA	MV_LJFINPRO
SIGATEC	MV_ATDIAS
SIGATMK	MV_TKCTILG

Nomeação de Tabelas

Tabela

Os nomes de novas Tabelas são disponibilizados pela Torre de Engenharia, dentro de um "*range*" para cada GDP. Assim cada nova tabela que uma GDP tiver necessidade de criar, tem que se orientar por esse "*Range*".

Exemplo:

GDP Venda & CRM: *Range* de MD1 até MD9

Além do "Range" de tabelas, é obrigatório que assim que definido quais tabelas, campos e índices que seram utilizados, sejam imediatamente cadastrado no AtuSx.

Recomendado também, que cada GDP faça um controle dessas tabelas, para saber quais estão sendo utilizadas e qual a próxima a ser utilizada.

TOTVS

Padrões de Design

Nomeação de Campos

XX[X]_YYYYYY[Y]

XX[X] - Segunda e terceira letra da tabela se a tabela começar com S ou as três letras da tabela se não começar com S.

YYYYYY[Y] - O nome representativo do campo com seis letras se a tabela não começar com S ou sete letras se a tabela começar com S.

Exemplo:

Tabela: MD1 - Processos

Campo: Código

Nome do campo: MD1_CODPRO

Tabela: SA1 - Clientes

Campo: Código da transportadora

Nome do campo: A1_TRANSP

TOTVS

Padrões de Design

Filial:

O campo que identifica a Filial do Registro, sempre deve ser preenchido com a palavra “FILIAL”, exemplo, MD1_FILIAL.

Outros campos:

Os nomes sempre tem que ser formado por mais de um sufixo, para ajudar a identificação dos campos.

MD1_CODIGO - ERRADO - POIS, ASSIM NÃO CONSEGUIMOS IDENTIFICAR “QUE CÓDIGO É”.

MD1_CODPRO - CERTO - POIS, ASSIM SABEMOS QUE É CÓDIGO DE PROCESSO.

Importante:

As informações no banco de dados não podem ser duplicadas, mesmo que a tabela seja de responsabilidade de outra GDP. O mesmo vale para os campos, pois o usuário não deve digitar duas vezes a mesma informação. Sempre que possível, consulte especialistas em outros módulos sobre se é possível encontrar a mesma informação em outra parte do sistema.

Nomeação de Pontos de Entrada

XXYYYZZZZ

XX - Iniciais do módulo.

YYY - Código sequencial do programa. Mantenha o padrão de numeração de 10 em 10.

ZZZZ - Nome dado ao ponto de entrada.

Exemplos:

Módulo	Parâmetro
SIGALOJA	LJ010ZZZZ
SIGATEC	AT010ZZZZ

TOTVS

Padrões de Design

Para criar e utilizar pontos de entrada, devemos ter em mente:

- Analisar o motivo da criação do ponto, pois é importante criá-lo num ponto que seja útil, não redundante e que atenda as condições do cliente;
- De forma alguma deve-se utilizar o ponto de entrada para corrigir eventuais falhas no sistema;
- Também não se devem inserir pontos de entrada em processo críticos do sistema, pois isto acarretará em resultados imprevisíveis;
- É imprescindível a utilização da função ExistBlock() que verifica a existência no ponto de entrada no repositório além de condicionar a execução do mesmo;
- Não tratar ponto de entrada com *find function* e *user function*;
- Não é necessário efetuar o *cache* da existência de um determinado ponto de entrada, a função *ExistBlock* já faz esse trabalho.

TOTVS

Padrões de Design

Exemplo:

```
// EntrancePointExample.prw
#INCLUDE "TOTVS.CH"

Function EntrancePointExample()

    While SA1->(!EOF())
        If ExistBlock("SAVECLI")
            ExecBlock("SAVECLI", .F., .F., aParam)
        EndIf
        SA1->(DbSkip())
    EndDo
Return
```

TOTVS

Estrutura de um programa Microsiga Protheus

Área de Cabeçalho

Na área de cabeçalho deve-se sempre informar os CHs utilizados no código-fonte, as variáveis estáticas, os comandos de pré-processamento específicos somente ao código-fonte e um comentário que explique e defina as responsabilidades e tarefas.

Não deve-se incluir arquivos de cabeçalho apenas por segurança. Se não se está referenciando nenhuma das constantes ou utilizando nenhum dos comandos contidos em um destes arquivos, a inclusão apenas tornará a compilação mais demorada.

O *include* “TOTVS.CH” é o *include* padrão da ADVPL e sempre deve ser informado.

TOTVS

Estrutura de um programa Microsiga Protheus

Área de Identificação

Esta é uma área dedicada a documentação do programa / função / método. Contém comentários explicando a sua finalidade, data de criação, parâmetros, retornos e alterações efetuados.

{Protheus.doc}	Identifica o inicio do bloco de documentação. Deve ser precedido do nome da função ou método e no parágrafo seguinte deve ter a descrição completa do bloco.
@param	Parâmetros de entrada, listados na ordem de passagem.
@protected	Se inserida indica que a função tem uso restrito pela GDP criadora e não pode ser reaproveitada em customizações e/ou integrações.
@author	Autor ou revisor do bloco
@version	Versão da Linha de produto Microsiga Protheus em que o bloco teve inicio. Utilize a nomenclatura definida pela GDP de *Framework*,.
@build	Versão mínima da *Build* do *Application Server* que o bloco é suportado.
@deprecated	Se inserida indica que a função não possui mais manutenção e/ou foi substituída por outra, tendo seu uso depreciado. Recomenda-se assim a substituição pela nova função.
@see	Indica as funções que devem ser observadas pelo desenvolvedor antes do uso. “Veja também”.
@since	Data de criação da rotina
@return	Indicador do retorno da função
@todo	Indicativo de função incompleta ou com pendências de desenvolvimento
@sample	Exemplo de uso do bloco
@obs	Observação complementar ao bloco.

TOTVS

Estrutura de um programa Microsiga Protheus

```
//-------------------------------------------------------------------
/*/{Protheus.doc} ComentariosMultiplasLinhas
Demonstra o uso de comentários em múltiplas linhas.

@sample    ComentariosMultiplasLinhas( nCat1, nCat2 )
@param     nCat1 - Cateto 1
           nCat2 - Cateto 2
@return    nHipo - Hipotenusa

@author    TOTVS
@since     12/06/2012
@version   P12
/*/
//-------------------------------------------------------------------
Function ComentariosMultiplasLinhas( nCat1, nCat2 )

//-------------------------------------
// Calcula a Hipotenusa de um triângulo
// retângulo de acordo com o teorema de
// Pitágoras onde, H² = C1² + C2².
//-------------------------------------
nHipo = SqRt( ( nCat1 ^ 2 ) + ( nCat2 ^ 2 ) )

Return nHipo
```

Estrutura de um programa Microsiga Protheus

Área de declaração de variáveis e ajustes iniciais

Nesta área devem ser feitos os ajustes iniciais, importantes para o correto funcionamento do programa. Entre esses ajustes iniciais se encontram declarações de variáveis, proteções e inicializações.

Exemplo de ajustes iniciais:

- GetArea()
- Definição de valor Default para parâmetros
- Abertura de Tabelas e Índices

Estrutura de um programa Microsiga Protheus

Corpo do Programa

É nesta área que se encontram as linhas de código do programa. É onde se realiza a tarefa necessária através da organização lógica destas linhas de comando. Espera-se que as linhas de comando estejam organizadas de tal modo que no final desta área o resultado esperado seja obtido, seja ele armazenado em um arquivo ou em variáveis de memória, pronto para ser exibido ao usuário através de um relatório ou na tela.

Área de Encerramento

É nesta área onde as finalizações são efetuadas e onde o resultado da execução do programa é utilizado. Pode-se exibir o resultado armazenado em uma variável ou em um arquivo ou simplesmente finalizar, caso a tarefa já tenha sido toda completada no corpo do programa.

Exemplo de ajustes de encerramento:

- RestArea()
- Fechamento de Arquivos
- Destruição de tabelas temporárias

Rotinas de Cadastro

O MVC foi criado com o objetivo de simplificar este processo, podendo ser utilizado para qualquer tipo de rotina de cadastro, seja ela simples ou complexa. Além disso, o MVC oferece uma série de vantagens de desempenho e usabilidade. Desta forma, recomenda-se fortemente o seu uso em todas as novas rotinas de cadastro.

Consulte a apostila de MVC em http://tdn.totvs.com/display/public/mp/AdvPl+utilizando+MVC para mais informações.

Rotinas de Consulta

Não há um padrão para a estrutura dos programas de consulta, dando ao desenvolvedor liberdade para utilizar a sua criatividade. Porém algumas regras devem ser seguidas:

- Consultas de formulário devem seguir o padrão dos programas de formulário;
- As *interfaces* de consulta devem ser rápidas;
- O uso do objeto *TreeView* deve ser evitado;
- Consultas que exportem dados devem seguir as regras de privilégio do cadastro de usuários;
- Caso a consulta manipule muitos dados, é preferível terceirizar esta responsabilidade para o banco de dados, utilizando-se de *queries* SQL.

Rotinas de Relatórios

Um relatório da linha Microsiga Protheus pode ser de dois tipos (Personalizáveis e Não personalizáveis), conforme a necessidade de impressão. Porém, os não personalizáveis devem ser utilizados apenas para manutenção do legado e em desenvolvimentos específicos, em que o cliente não pode alterar o leiaute do relatório, como os relatórios legais (DANFe, P1/P2/P8/P9, Diário Geral, Razão Contábil, etc.).

Localização , tradução e Internacionalização

Tradução

A preparação para tradução dos programas da linha Microsiga Protheus é realizada através dos *header´s* (CH) de programa. Por padrão os *header´s* de tradução devem ter o mesmo nome do programa, exemplo: MATA410.PRX o header de tradução deste programa será o MATA410.CH

- O desenvolvedor deve localizar as *STRING´s* do programa e substituí-las pelo DEFINE;
- As *STRING´s* utilizadas em combos ou listas devem possuir o código numérico, em vez de letras. Exemplo: em vez de S-Sim;N-Não, utilize 1-Sim;2-Não.
- O arquivo de *header* deve ser incluído na ferramenta AtuSx;
- As *string´s* podem conter palavras ou frases;
- Para frases móveis, deve-se utilizar a função I18N. I18N("A casa de #1[nome de pessoa]# é #2[cor]#",{cNome,cCor})
- Não se deve incluir formatação de números e datas na tradução;
- Não se deve inferir nenhuma tradução diretamente no código fonte;
- Não se deve incluir textos em imagens;
- Recomenda-se que o dado apresentado em tela, proveniente do dicionário de dados seja obtido dele.

Internacionalização

A internacionalização da linha Microsiga Protheus é realizada pelo parâmetro de sistema (SX6) MV_PAISLOC. Este parâmetro indica qual país o dicionário de dados provém e com base nele fé efetuada a separação dos blocos de código internacionalizado.

```
Do Case

	Case SuperGetMV("MV_PAISLOC") == "BRA"
		//Calculo do ICMS

	Case SuperGetMV("MV_PAISLOC") == "ARG"
		//Calculo do IVA

	OtherWise
		//Calculo do IVA

EndCase
```

TOTVS

Trabalhando com dados

- Ao utilizar variáveis de transporte de dados de campos, esta deve ter o tamanho respeitado;
- Utilizar GetArea() e RestArea() para guardar e restaurar o posicionamento de registros;
- Ao utilizar o DbSeek(), verifique sempre se o registro foi encontrado;
- O uso do DbGoTop() e DbSeek() apenas com a filial é restrito;
- A função Posicione não restaura a posição original;
- Utilize a função ExistCpo() para verificar se existe um registro em outra tabela;
- Para travar um novo registro ou a edição dele, use a função RecLock() e para liberar MsUnlock();

TOTVS

Trabalhando com dados

Não é recomendado o uso das seguintes funções para efetuar travamentos e destravamentos:

- DBRLock;
- DBRUnlock;
- MSRLock;
- MSRUnlock;
- DBUnlock;
- DBUnlockAll;
- MultLock(Alias, aChaves, nOrd);
- SoftLock;
- MSUnlockAll;
- MSUnlockSoft.

TOTVS

Trabalhando com dados

Leitura

O nível de isolamento é Read Uncommitted, por isso utilize o Reclock() para evitar que duas transações gravem o mesmo registro;

Bloqueio de Interface

Utilize a função SoftLock() para bloquear um registro quando não precisar gravá-lo. Seu uso não substitui o RecLock().

Bloqueio de Processamento

Caso seja necessário fazer um bloqueio utilize semáforos, LockByName é o mais recomendado, mas seu uso deve ser evitado;

Utilze Begin Transaction, DisarmTransaction() e End Transaction para controlar o início e o fim de uma transação com mais de uma tabela.

Veja mais em: http://tdn.totvs.com/pages/viewpage.action?pageId=22480734

TOTVS

Trabalhando com arquivos

- Arquivos e índices temporários devem ser utilizados com cuidado, pois podem gerar um tempo de resposta longo enquanto estão sendo construídos. Ao término devem ser apagados.
- Utilize a função GetDbExtension() para retornar a extensão do arquivo de trabalho.
- Ao utilizar filtros estes devem ser desabilitados.
- Ao fazer Queries, utilize comandos padrão ANSI, aceitos por qualquer banco de dados suportado.
- Recomenda-se que novas queries devem ser escritas utilizando Embedded SQL

TOTVS

Legado

- Proteger novos campos com FieldPos();
- Proteger novas tabelas com AliasInDic();
- Utilize a função AjustaSX1 para criar perguntas novas quando necessário;
- Utilizar valores padrão nas funções GetMV() e SuperGetMV();
- Em caso de inclusão de uma chamada a uma nova função, utilize Findfunction() para proteger;
- Em caso de inclusão de um método na classe, proteger com MethIsMemberOfSample();
- Em caso de inclusão de uma propriedade na classe, proteger com AttIsMemberOfsample();

Quando há uma virada de versão onde os fontes são separados, recomenda-se fazer uma "limpeza", removendo funções de proteção e funções de ajuste;

TOTVS

Proteção contra erros

- Utilizar as funções de controle para o legado;
- Tratar tipos de variáveis, usando Type() ou ValType();
- Utilize a função Len() para verificar o tamanho de um Array;
- Defina valores padrão para parâmetros de uma função.

TOTVS

Preferência por comandos ao invés de construtor de classe

Sempre que possível, dar preferência ao uso do comando ao invés da utilização de construtores ou acesso a métodos, por exemplo para a classe *TSay*:

```
#INCLUDE "TOTVS.CH"

Function CommandVsConstructor()

Local oDlg  := Nil
Local oSay  := Nil

Define Dialog oDlg Title "Command Vs Constructor" From 0,0 To 300,300 Pixel

// Versão utilizando o construtor
oSay:= TSay():New(05, 05, {||"Texto para exibição"}, oDlg,,,,,,.T., CLR_RED, CLR_WHITE, 200,  20)

// Versão utilizando o comando
@15,05 Say oSay Prompt "Texto para exibição" Of oDlg Pixel Colors CLR_RED,CLR_WHITE Size 200,20 )

Activate Dialog oDlg Centered

Return
```

TOTVS

Desempenho

- O uso intenso de refresh pode causar uma queda no desempenho da rotina;
- Eval tem um desempenho inferior ao uso de & em macro execução;
- MsSeek() pode ser mais rápido do que o DbSeek() em casos de grande número de posicionamentos;
- SuperGetMV() permite guardar um parâmetro em cache, evitando uma nova consulta ao dicionário, portanto seu desempenho é melhor que o GetMV();
- ValType() é mais rápida que a função Type(), mas deve ser utilizada apenas se a variável existir;

Variáveis

- Não é recomendável utilizar múltiplas atribuições;
- Cuidado com o uso de variáveis que recebem dados de diferentes tipos;
- Variáveis públicas são acessadas de qualquer parte do sistema, dificultam os testes e ocupam memória desnecessária;
- O uso de parênteses facilita a leitura. Ex. Local nValor := (2 + (10/2) + (5*3) + (2^3))
- Em caso de operações ternárias utilize parênteses. Ex. lOk := IIf((nPeriod == "1"), .T., .F.)
- O uso de constantes facilita o entendimento do código;

Funções

- Crie funções curtas, evite as funções Bombril;
- Classes base não devem depender de suas derivadas;
- Mantenha dados de configuração nos níveis mais altos;
- Classes que não tenham uma real dependência não devem ser acopladas artificialmente.
- Utilizar funções genéricas ao invés de criar novas funções;
- Manter o fluxo da função ordenado;
- Evite duplicidade de código criando funções genéricas;
- Não utilizar mais de um Return ou declarações GoTo;
- Descartar funções mortas;
- Deixe-a estéticamente bonita;

Nomenclatura

O nome deve simplesmente dizer o que a variável, método ou classe é. E a variável, método ou classe deve ser simplesmente o que o nome diz.

Comentários

- Deve-se evitar comentários que não acrescentam nenhuma informação ou não ajudam a entender o código;
- O comentário serve para ajudar o leitor a saber tanto quanto o autor do código no momento de sua criação;
- Comentários devem ter o máximo de informação no menos espaço possível;
- Evite palavras ambíguas;
- Quando um código deixar de ser utilizado, ele deve ser deletado e não comentado;
- Não comentar fim de estruturas como If, While, etc. Essa prática pode até parecer útil, mas na maioria dos casos isso costuma ser comentários pra maquiar um código ruim;
- Quando um código for alterado e seu comentário ficar obsoleto, vago até mesmo errado, é fundamental alterar, remover ou adicionar o que for pertinente para manter o comentário compatível com o código que ele se refere;
- Alguns comentários padrões costumam ser usados por grande parte da comunidade de desenvolvedores e seus significados são padrões. ToDo, FixMe, Hack e XXX.

Simplificando laços e lógica

- Simplifique a condição para execução de blocos condicionais;
- Para deixar a leitura mais natural, mantém-se na esquerda o argumento que será questionado;
- Evite expressões condicionais na negativa;
- Utilizar o *IF* ternário apenas em situações simples, onde não há expressões longas, mas apenas uma comparação e atribuições simples e objetivas;
- Utilizar variáveis para quebrar expressões muito grandes;
- Use a lei de Morgan para distribuir o "*not*" ou "!" em expressões booleanas. Ex:
- !(lVarA .And. lVarB .And. lVarC) => (!lVarA) .Or. (!lVarB) .Or. (!lVarC)

TOTVS

Saiba mais

TDN – TOTVS Developer Network: http://tdn.totvs.com
Boas Práticas de Programação: http://tdn.totvs.com/pages/viewpage.action?pageId=22480352

TOTVS

Questões

Danilo Dias
GDP de Serviços

danilo.dias@totvs.com.br