# Manual de Desenvolvimento

## CODE CONVENTIONS (v1)

Convenções básicas de programação **pVISTA**
METRO VERSION

**PROEVENTO** T E C N O L O G I A
Equipe de Desenvolvimento

As definições gerais sobre a nomes das classes css a serem utilizadas correspondentes a PROEVENTO MetroUI pode ser consultada online em
**http://virtualboss.proevento.com.br/_metrodocs/**

Para referência prática sobre codificação e padrões (html/css, vbscripts/javascript) convencionados neste manual, o módulo **_pvista/modulo_CfgPanel** do sistema pVISTA deve ser utilizado como modelo.

# Introdução

Aos amigos desenvolvedores, apenas uma ponderação que escrevi a partir de perguntas ao logo dos anos treinando, ensinando, aprendendo e trabalhando com vários programadores e amigos.

Realmente acredito que o PONTO-CHAVE de tudo isso, ou ao menos para escrever um bom código, você deve se importar com ele, digo, para ser um programador melhor, é necessário investir tempo e esforço neste sentido.

Uma boa programação não nasce somente da competência técnica. Já vi programadores altamente intelectuais que conseguem produzir algoritmos intensos e impressionantes, conhecem o padrão de sua linguagem de cor, mas escrevem o pior código possível. É um código difícil de ler, de usar e de modificar. Já vi programadores mais humildes que se atêm a um código bem simples, porém escrevem programas elegantes e expressivos com os quais é uma satisfação trabalhar.

Com base nestes meus já 30 anos de programação e de experiência na produção de software, concluí que a verdadeira diferença entre programadores medíocres e programadores excelentes é esta: ATITUDE. Uma boa programação resulta da adoção de uma abordagem profissional e de querer escrever o melhor software que você puder, levando em conta as limitações e as pressões do mundo real.

A experiência me trouxe a convicção de que um dos itens de extrema importância a influenciar na qualidade final de um software é a sua padronização de código fonte. Podemos pensar em padrões existentes, adaptar algum às nossas próprias necessidades ou simplesmente criar o nosso próprio padrão. Amigos, este documento (em sua primeira versão) busca incentivá-los a manter, criar e desenvolver em conjunto, padrões e um certo “gosto” por eles.

Seja qual for o padrão, **SE** ele contemplar ser: 1 – de nomes; 2 – de comentários; 3 – do tamanho das linhas; 4 – da denominação de variáveis, classes, funções; 5 – das estruturas de decisão e de repetição; 6 – da quantidade de linhas em branco deixadas entre os segmentos; etc. **ENTÃO** este será o padrão ideal, ou estará muito próximo disso.

As diferenças no estilo de programação dificultam a compreensão do código pelos membros da equipe e, consequentemente, a manutenção deste código também será comprometida. Procure adotar um padrão que seja fácil e simples. Ao identificar um código fora do padrão alerte toda a equipe. Coloque este código no padrão. Publique as regras do padrão que geram mais dúvidas.

Ajuste o padrão para que ele se aproxime da realidade dos programadores. Com certeza o software irá ganhar um pouco mais de qualidade. Pessoal, sugiro fortemente: adote um padrão de código onde quer que você esteja.

Alessander Oliveira

# Boas práticas

"*Os futuros programadores só terão contato com uma padronização de código no mercado de trabalho, isto se empresa utilizar tal artefato. Infelizmente, eles não levam esta prática para o mercado, pois a maioria não terá um contato direto com um padrão de código nos bancos universitários. A padronização de código é apresentada aos alunos, geralmente pelos professores da área engenharia de software, de maneira isolada. Infelizmente este conceito não é disseminado pelas disciplinas de Linguagem Técnica de Programação, Estrutura de Dados, Programação Avançada, Programação Orientada a Objetos, etc.*" - José A. Fabri

## Bibliotecas / Includes

Ao incluir trechos de código não utilizamos a prática de include "virtual", apenas "file" sendo para este aplicada a lógica de caminho relativo: Ex. <!--#include file="../../ajax/planilha.asp"-->

Como boa prática, agrupamos todos os includes no topo de nossa página com exceções devidamente justificadas caso a caso.

## Padrão de cabeçalho

Utilizar o mesmo formato de cabeçalho para iniciar os códigos ASP. È desejável logo após os includes que algumas informações sejam declaradas e mantidas. Um exemplo de cabeçalho a ser seguido e mantido:

```
<%
' ------------------------------------------------------------
' Página....: CadastrarNononono.asp
' Criação...: 20/10/2014 (Mauro/Aless)
' ------------------------------------------------------------
' Definição:
'
' ------------------------------------------------------------
' DD/MM/YYYY | PESSOA     | DESCRIÇÃO DA ALTERAÇÃO
' ------------------------------------------------------------
' 18/08/2014 | Alan/Aless | Inserimos o comentário de exemplo
'                           para logica de exportação
' 27/02/2015 | Eli/Gabi   | Segundo exemplo sobre retorno
' ------------------------------------------------------------
%>
```

## Usar sempre “Option Explicit”

Sempre utilizar a opção explicita de declaração de variáveis, desta forma fica obrigado a utilização do Dim e Const: Option Explicit

* No caso deste projeto, existem includes como o athDBConnCS que já determina “Option Explicit“, portanto observar os casos onde deve ser necessário.

## Identificar tipo de variáveis

No ASP não é definido o tipo da variável, mas para facilitar, devemos procurar inserir o tipo antes do nome da variável SEMPRE que POSSÌVEL. Com isto evitaremos confusão na hora de instanciar valores:

```
<%
Const CFG_DBDADOS = “xmandata”
Const CFG_CPF     = “90290016055”


Dim strNome         ' String
Dim intConta        ' Inteiro
Dim blnVerifica     ' Boleano
Dim objConn,objRS   ' Object
Dim arrCar,matCont ' Colections
%>
```

Para declaração de constantes utilize letras MAIÚSCULAS.

Em alguns casos, mesmo que não identificado o tipo, podemos declarar variáveis de acordo com ao que se destinam. Tendo em mente que a ideia principal é facilitar leitura e manutenção do código. Por exemplo: flgAutoriza, auxCod.

* Declarar variáveis com uma única letra (i, j, k, l, x, y, z), apenas para contagens/indexações simples e num escopo de tamanho controlado/isolado (pequenos laços de for, etc...):. Prefira nomes como “Cont”, “ContPagina”, etc quando o escopo é mais abrangente e se faz necessário o maior entendimento de sua funcionalidade.

– Escrevendo de outra MANEIRA: Se você vai usar uma variável para incremento, contador ou só para armazenar um valor que será usado nas próximas linhas, ainda mais quando se trata de valores inteiros, não há necessidade de usar nomes grandes e descritivos… É muito comum ( em nosso caso um padrão) usar i, j, k, l, x, y, z para contadores em for/while, por exemplo.

## Instancia de Objetos e coleções:

Ao setar um novo objeto ou carregar um RecordSet com uma coleção de dados, devemos instanciar seus valores em variáveis locais e logo desalocar estes espaços na memória:

```
<%
Dim objConn, objRS
Dim strDado

VarLocal = ""

Set objConn = Server.CreateObject("...")
Set objRS = objConn.Open ("SELECT * FROM TABLE")

If Not objRS.Eof Then
   strDado = getValue(objRS,"campo01")
End If

Set objRS = Nothing
objConn.close
%>
```

ATENÇÃO:

Além disso, utilizar sempre a função getValue ([Rset],[Nfield]) para pegar dados de um recordset. Exemplo: onde usar-se-ia objRS("campo01"), deve ser utilizado **getValue(objRS,"campo1")**

## Indentação:

A indentação é essencial para se ter um código fonte de fácil leitura e rápido entendimento.

Muitos profissionais desaconselham a utilização da tecla TAB para a isso, devido às várias interpretações possíveis por parte de editores de código, ou seja, alguns editores interpretam um TAB como sendo 4 espaços, outros como sendo 8 espaços.

O mais importante é manter um padrão de identação que facilite a identificação de estruturas hierarquizadas desde um simples IF THEN ELSE END IF, até Tags DIV, TABLE compostas.

* O Dreamweaver possui o comando: Commands > Apply Source Formatting, que funci0ona bem para códigos "puro" (de uma linguagem somente), mas quando mesclamos html + asp, por exemplo a indentação que ele produz deixa muito a desejar.

## Submeter um Formulário:

Regra básica, não submeter um formulário usando recurso do HTML, usar sempre javascript como no exemplo abaixo:

ERRADO:

```
<body>
 <form action="pagina_destino.asp" method="post">
 <input type="submit" id="btnLiberar" value="Salvar" />
</form>
```

**CERTO:**

```
<head>
<script type="text/javascript" language="JavaScript">
function submeter() {
 if (Faz_a_validação_dos_campos(Formulario)) {
    Formulario.action = "pagina_destino.asp";
    Formulario.method = "post";
    Formulario.submit();
 }
}
</script>
</head>
<body>
 <form action="#" method="POST">
 <input type="button" id="btnLib" onclick="submeter();"/>
 </form>
</body>
```

## Tratamento de Erros:

Outra Regra básica, procurar não usar o “On Error Resume Next” quando não houver tratamento de erro, pois fica sem sentido na página (salvo exceções), pois apenas esconde o erro e gera inconsistência de dados, exibindo informações equivocadas para o usuário, já que o erro não esta sendo tratado como deveria. Prática correta:

```
<%
On Error Resume Next

If Err.number > 0 Then
  Trata o erro e exibe de forma amigável para o usuário.
End If
%>
```

## Princípios DRY, DIE e KISS

Ao criar seu código tenha isso esses princípios em mente e, busque entendê-los, compreendê-los e usá-los, e irá notar uma incrível melhoria na qualidade e eficiência do seu código.

**DRY** ou Don’t Repeat Yourself (Não Se Repita) - Baseia-se no conceito de que computadores e sistemas são feitos da automação de tarefas repetitivas e o seu código não deve ser diferente. Você não deve duplicar uma linha de código pra fazer a mesma coisa!

**DIE** ou Duplication Is Evil (Duplicação é malígna/má) - Segue o mesmo conceito do DRY.

**KISS** ou Keep It Simple, Stupid (Mantenha-o simples, estúpido) - Determina que quanto mais simples e enxuto for seu código, melhor. Simples e rápido.

# Especificidades

PRINCIPAL

- Dentro do sistema em _pvista/ compreendemos como uma abstração de “modulo”, topdas as pastas denominadas: /modulo_[nomedomodulo];
- Na pasta do módulo já criada o arquivo principal será o “**default.asp**”;
    - Esta página basicamente é formada por dois includes onde um carregara o filtro e o outro a grade **_include_filtro.asp** e **_include_grade.asp**)
    - Todas variáveis são declaradas na default, evitar declarações nos includes.
    - Observa que a consulta principal(que alimenta a grade) é definida partir dos parâmetros de filtragem na parte anterior ao <body>. Estruturas de arrays são utilizadas para exibição dos e setup adequado da tablesort.
- Constantes:
    - Const MDL = "DEFAULT" ' - Default do Modulo...
    - Const LTB = "sys_tabela" ' - Nome da Tabela...
    - Const DKN = "cod_chavel" ' - Campo chave...
    - Const DLD = "../modulo_Padrao/default.asp" ' "../evento/data.asp" - 'Default local após Deleção
    - Const TIT = "Nomequecarreganofiltro"

GERAIS

- Evitar inserção de componentes fora da classe metro e quando necessário comentar código
- Includes(DB,Script,Secury): includes utilizados no padrão metro. Importante a inserção deles pois a metro não funcionar sem esse includes no tipo da página.
    - <!--#include file="../_database/athdbConnCS.asp"-->
    - <!--#include file="../_database/athUtilsCS.asp"-->
    - <!--#include file="../_database/secure.asp"-->
    - <!--#include file="../_metroui/meta_css_js.inc"--> (dentro do HEAD no HTML)
    - <script src="../_scripts/scriptsCS.js"></script> (dentro do HEAD no HTML)
- ATENÇÃO: doctype, language, option explicit, etc... estão no athDBConn
- getParam(no lugar de Request), getValue(no lugar da objRS sozinha);
- Não há um padrão rígido quanto a criação dos nome de variáveis entretanto devemos seguir os exemplos do modulo que serve de modelo. Abaixo seguem algumas boas praticas desejaveis:.
    - Respeitar os padrões de variáveis de formulário que farão uso das *ToDB (insToDB...), ex.: DBVAR_STR_TEXTO (Prefixo_TipoCampoTabela_NomeCampoTabela)
    - Variáveis que servem para repasse de valores entre formulários (post/get, GetParam), utilizar prefixo “var_ “
- Use como modelo o módulo... CFG PANEL(http://192.168.1.8:83/_pvista/modulo_CFGPanel/);
- Verboses não contem botões de ação fora as exceções;
- Dialogs tem cores padrões que devem ser respeitas e inseridas através da classe;