```
Option Explicit

Dim lngmEndProcLine As Long
Dim lngmSCol As Long
Dim lngmELine As Long
Dim lngmECol As Long
Dim vbpmProject As VBIDE.VBProject
Dim vbcmComponent As VBIDE.VBComponent
Dim vbcmmCodeModule As VBIDE.CodeModule
Dim vbcpmCodePane As VBIDE.CodePane
Dim smProjectName As String
Dim smModuleName As String
Dim smProcedureName As String
Dim lngmCurrentModuleLineNum As Long
Dim lngmModuleProcStartLine As Long
Dim lngmModuleProcBodyLine As Long
Dim lngmProcCountLines As Long
Dim lngmModuleProcEndLine As Long
Dim smCurrentLineText As String
Dim smSelection As String
Dim smCodeArray() As String
Dim lngmFirstDimLine As Long
Dim lngmLastDimLinePlus1 As Long
Dim lngmLineAfterDefinitionPlus1 As Long
Dim blnmNonConsecutiveDims As Boolean
Dim smDimsArray() As String
Dim lngmDimCount As Long
Dim lngmMaxDepth As Long
Dim lngmBlankLines As Long
Dim lngmCodeLines As Long
Dim lngmCommentLines As Long
Dim lngmTotalOperators As Long
Dim lngmUniqueOperators As Long
Dim lngmTotalOperandsCount As Long
Dim lngmUniqueOperandsCount As Long
Dim lngmLongLines As Long
Dim lngmParameterCount As Long
Dim smParameterArray() As String
Dim smLocalVariablesArray() As String
Dim lngmLocalVariablesCount As Long
Dim smFirstIndent As String


```

```
Private Sub Class_Initialize()
' I'm aware that this procedure performs multiple passes
'  through the code of a procedure and that some of
'  these could be merged.
'
' The multi pass approach is delibarate.
' Each of the sections can be worked on without
'  affecting the others and I think the
'  processing is clearer.
'
' There are a LOT of Dims here.
' This is because the class reports a LOT of
'  information.
'
' Assumes ONE Dim/Const/Static per line.
'

Dim blnlNonConsecutiveDims As Boolean
Dim lnglArrayIndex As Long
Dim lnglBlankLines As Long
Dim lnglCLine As Long
Dim lnglCodeLines As Long
Dim lnglCommentLines As Long
Dim lnglCurrentModuleLineNum As Long
Dim lnglDimCount As Long
Dim lnglECol As Long
Dim lnglELine As Long
Dim lnglEndProcLine As Long
Dim lnglFirstDimLine As String
Dim lnglLastDimLinePlus1 As Long
Dim lnglLineAfterDefinitionPlus1 As Long
Dim lnglLongLines As Long
Dim lnglMaxDepth As Long
Dim lnglModuleProcBodyLine As Long
Dim lnglModuleProcEndLine As Long
Dim lnglModuleProcStartLine As Long
Dim lnglProcCountLines As Long
Dim lnglSCol As Long
Dim lnglSLine As Long
Dim lnglTotalOperandsCount As Long
Dim lnglTotalOperators As Long
Dim lnglUniqueOperandsCount As Long
```

```
    Dim lnglUniqueOperators As Long
    Dim slBranch As String
    Dim slBranchArray() As String
    Dim slBranchEnd As String
    Dim slBranchEndArray() As String
    Dim slCn As String
    Dim slCodeArray() As String
    Dim slCodeLines As String
    Dim slCurrentLineText As String
    Dim slDefinitionLine As String
    Dim slDimsArray() As String
    Dim slDQ As String
    Dim slJoinedCodeArray() As String
    Dim slLine As String
    Dim slLineArray() As String
    Dim slModuleName As String
    Dim slOperandsArray() As String
    Dim slOperatorArray() As String
    Dim slOperators As String
    Dim slProcedureName As String
    Dim slProjectName As String
    Dim slSelection As String
    Dim slTrimLine As String
    Dim slLocalVariablesArray() As String
    Dim vbclComponent As VBIDE.VBComponent
    Dim vbcmlCodeModule As VBIDE.CodeModule
    Dim vbcplCodePane As VBIDE.CodePane
    Dim vbplProject As VBIDE.VBProject
    Dim lnglParameterCount As Long
    Dim lnglB1Pos As Long
    Dim lnglB2Pos As Long
    Dim slParamsArray() As String
    Dim lnglN As Long
    Dim slPVariablesArray() As String
    Dim slDVariablesArray() As String
    Dim lnglLocalVariablesCount As Long
    Dim slfirstindent As String

    slDQ = Chr(34)
    slCn = " _"
    slOperators = "=,>,<,>=,<=,<>,And,Or,Xor,Not,+,-,*,/,\,^,Mod,&"
    slOperatorArray = Split(slOperators, ",")
    slBranch = "If,ElseIf,Do,Select Case,Case,For"
```

```
    slBranchEnd = "End If,End If,Loop,End Select,End Select,Next"

    ' ----------------------------------------------------------------
    ' Get project objects.

    Set vbplProject = Application.VBE.ActiveVBProject
    slProjectName = vbplProject.Name

    Set vbcplCodePane = Application.VBE.ActiveCodePane

    Set vbcmlCodeModule = vbcplCodePane.CodeModule
    slModuleName = vbcmlCodeModule.Name
    slModuleName = vbcplCodePane.CodeModule.Parent.Name

    Set vbclComponent = vbplProject.VBComponents(slModuleName)

    vbcplCodePane.GetSelection lnglCurrentModuleLineNum, lnglSCol, lnglELine, lnglECol

    ' ----------------------------------------------------------------
    ' Get some code items from project objects.

    slProcedureName = vbcmlCodeModule.ProcOfLine(lnglCurrentModuleLineNum, vbext_pk_Proc)
    lnglModuleProcStartLine = vbcmlCodeModule.ProcStartLine(slProcedureName, vbext_pk_Proc)
    lnglModuleProcBodyLine = vbcmlCodeModule.ProcBodyLine(slProcedureName, vbext_pk_Proc)
    lnglProcCountLines = vbcmlCodeModule.ProcCountLines(slProcedureName, vbext_pk_Proc)
    lnglModuleProcEndLine = lnglModuleProcStartLine + lnglProcCountLines - 1

    ' ----------------------------------------------------------------
    ' Current Line.

    slLine = vbcmlCodeModule.Lines(lnglCurrentModuleLineNum, 1)
    slTrimLine = Trim$(slLine)
    slCurrentLineText = slLine

    ' ----------------------------------------------------------------
    ' Get selection.
    ' Set Indent.

    If lnglELine - lnglCurrentModuleLineNum = 0 Then
      If lnglECol - lnglSCol = 0 Then

        ' No Selection.
        slSelection = ""
```

```
    slfirstindent = ""

  Else

    ' Selection on this line.
    slSelection = Mid$(slLine, lnglSCol, lnglECol - lnglSCol)
    slfirstindent = ""
    For lnglN = 1 To Len(slSelection)
      If Mid$(slSelection, lnglN, 1) = " " Then
        slfirstindent = slfirstindent & " "
      Else
        Exit For
      End If

  End If

Else

  slSelection = vbcmlCodeModule.Lines(lnglCurrentModuleLineNum, lnglELine - lnglCurrentModuleLineNum + 1

End If

' ----------------------------------------------------------------
' Get First Dim line.

lnglFirstDimLine = 0
lnglCLine = lnglModuleProcBodyLine
Do

  slLine = Trim$(vbcmlCodeModule.Lines(lnglCLine, 1))

  If Left$(slLine, 4) = "Dim " Then
    lnglFirstDimLine = lnglCLine
    Exit Do
  ElseIf Left$(slLine, 6) = "Const " Then
    lnglFirstDimLine = lnglCLine
    Exit Do
  ElseIf Left$(slLine, 7) = "Static " Then
    lnglFirstDimLine = lnglCLine
    Exit Do
  End If

  lnglCLine = lnglCLine + 1
```

```
  If lnglCLine >= lnglModuleProcEndLine Then
    Exit Do
  End If
Loop

' ------------------------------------------------------------------
' Get Last Dim line plus 1.

lnglLastDimLinePlus1 = 0
lnglCLine = lnglModuleProcEndLine
Do

  slLine = Trim$(vbcmlCodeModule.Lines(lnglCLine, 1))

  If Left$(slLine, 4) = "Dim " Then
    lnglLastDimLinePlus1 = lnglCLine + 1
    Exit Do
  ElseIf Left$(slLine, 6) = "Const " Then
    lnglLastDimLinePlus1 = lnglCLine + 1
    Exit Do
  ElseIf Left$(slLine, 7) = "Static " Then
    lnglLastDimLinePlus1 = lnglCLine + 1
    Exit Do
  End If

  lnglCLine = lnglCLine - 1

  If lnglCLine <= lnglModuleProcBodyLine Then
    Exit Do
  End If
Loop

' ------------------------------------------------------------------
' Get the first line AFTER the definition.

lnglLineAfterDefinitionPlus1 = 0
lnglCLine = lnglModuleProcBodyLine
Do

  slLine = Trim$(vbcmlCodeModule.Lines(lnglCLine, 1))

  If Right$(slLine, 1) <> "_" Then
```

```
    If lnglCLine = lnglModuleProcBodyLine Then
      lnglLineAfterDefinitionPlus1 = lnglCLine + 1
    Else
      lnglLineAfterDefinitionPlus1 = lnglCLine
    End If
    Exit Do
  End If

  lnglCLine = lnglCLine + 1

Loop

' ---------------------------------------------------------------------
' Are there any Dims in the middle of the code?

blnlNonConsecutiveDims = False
If lnglLastDimLinePlus1 = 0 Then
  lnglCLine = lnglModuleProcBodyLine
Else
  lnglCLine = lnglLastDimLinePlus1 - 1
End If

Do

  slLine = Trim$(vbcmlCodeModule.Lines(lnglCLine, 1))

  If Left$(slLine, 4) = "Dim " Then
  ElseIf Left$(slLine, 6) = "Const " Then
  ElseIf Left$(slLine, 7) = "Static " Then
  ElseIf Left$(slLine, 1) = "'" Then
  ElseIf Len(slLine) = 0 Then

  Else
    blnlNonConsecutiveDims = True
    Exit Do
  End If

  lnglCLine = lnglCLine - 1

  If lnglCLine <= lnglLineAfterDefinitionPlus1 - 1 Then
    Exit Do
  End If
Loop
```

```
' ------------------------------------------------------------------
' Count All of the Dims and Parameters.

lnglDimCount = 0
If lnglLastDimLinePlus1 = 0 Then
  lnglCLine = lnglModuleProcBodyLine
Else
  lnglCLine = lnglLastDimLinePlus1 - 1
End If

' Count Dims.
Do

  slLine = Trim$(vbcmlCodeModule.Lines(lnglCLine, 1))
  If LenB(slLine) <> 0 Then
    slLineArray = Split(slLine, " ")
    Select Case slLineArray(0)
    Case "Dim", "Const", "Static"
      lnglDimCount = lnglDimCount + 1
    End Select
  End If

  lnglCLine = lnglCLine - 1

  If lnglCLine <= lnglLineAfterDefinitionPlus1 - 1 Then
    Exit Do
  End If

Loop

' ------------------------------------------------------------------
' Populate the DIMS array.

If lnglDimCount > 0 Then
  ReDim slDimsArray(lnglDimCount - 1)

  If lnglLastDimLinePlus1 = 0 Then
    lnglCLine = lnglModuleProcBodyLine
  Else
    lnglCLine = lnglLastDimLinePlus1 - 1
  End If

```

```
    lnglDimCount = -1
    Do

      slLine = Trim$(vbcmlCodeModule.Lines(lnglCLine, 1))
      If LenB(slLine) <> 0 Then
        slLineArray = Split(slLine, " ")
        Select Case slLineArray(0)
        Case "Dim", "Const", "Static"
          lnglDimCount = lnglDimCount + 1
          slDimsArray(lnglDimCount) = slLine
        End Select
      End If

      lnglCLine = lnglCLine - 1

      If lnglCLine <= lnglLineAfterDefinitionPlus1 - 1 Then
        Exit Do
      End If

    Loop

    'ReDim slDVariablesArray(UBound(slDimsArray))
    slDVariablesArray = slDimsArray

    For lnglN = 0 To UBound(slDVariablesArray)
      slDVariablesArray(lnglN) = Replace(slDVariablesArray(lnglN), "Dim ", "")
      slDVariablesArray(lnglN) = Replace(slDVariablesArray(lnglN), "Static ", "")
      slDVariablesArray(lnglN) = Replace(slDVariablesArray(lnglN), "Const ", "")
    Next lnglN
    For lnglN = 0 To UBound(slDVariablesArray)
      slLine = slDVariablesArray(lnglN)
      lnglB1Pos = InStr(slLine, " As ")
      If lnglB1Pos > 0 Then
        slDVariablesArray(lnglN) = Mid$(slLine, 1, lnglB1Pos - 1)
      End If
    Next lnglN
    For lnglN = 0 To UBound(slDVariablesArray)
      slLine = slDVariablesArray(lnglN)
      lnglB1Pos = InStr(slLine, " = ")
      If lnglB1Pos > 0 Then
        slDVariablesArray(lnglN) = Mid$(slLine, 1, lnglB1Pos - 1)
      End If
    Next lnglN
```

```
Else
  ' No Dims.
    lnglDimCount = -1
End If

lnglDimCount = lnglDimCount + 1

' --------------------------------------------------------------------
' Get the Parameters.

lnglCLine = lnglModuleProcBodyLine
slDefinitionLine = ""
Do

  slLine = Trim$(vbcmlCodeModule.Lines(lnglCLine, 1))
  slDefinitionLine = slDefinitionLine & slLine

  If Right$(slLine, 1) = "_" Then
    lnglCLine = lnglCLine + 1
  Else
    Exit Do
  End If

Loop

lnglB1Pos = InStr(slDefinitionLine, "(")
lnglB2Pos = InStrRev(slDefinitionLine, ")")
If lnglB2Pos - lnglB1Pos = 1 Then
  lnglParameterCount = 0
Else

  slDefinitionLine = (Trim$(Mid$(slDefinitionLine, lnglB1Pos + 1, lnglB2Pos - lnglB1Pos - 1)))
  slDefinitionLine = Replace(slDefinitionLine, " _", "")
  slDefinitionLine = Replace(slDefinitionLine, " As ", "@")
  slDefinitionLine = Replace(slDefinitionLine, "Optional ", "}")
  slDefinitionLine = Replace(slDefinitionLine, "ByRef ", "}")
  slDefinitionLine = Replace(slDefinitionLine, "ByVal ", "}")
  slDefinitionLine = Replace(slDefinitionLine, "ParamArray ", "}")
  slDefinitionLine = Replace(slDefinitionLine, " ", "")
  slParamsArray = Split(slDefinitionLine, ",")
  lnglParameterCount = UBound(slParamsArray) + 1
```

```
    slPVariablesArray = Split(slDefinitionLine, ",")

    For lnglN = 0 To UBound(slPVariablesArray)
      slLine = slPVariablesArray(lnglN)
      lnglB1Pos = InStr(slLine, "@")
      If lnglB1Pos > 0 Then
        slPVariablesArray(lnglN) = Mid$(slLine, 1, lnglB1Pos - 1)
      End If
    Next lnglN
    For lnglN = 0 To UBound(slPVariablesArray)
      slLine = slPVariablesArray(lnglN)
      lnglB1Pos = InStr(slLine, "=")
      If lnglB1Pos > 0 Then
        slPVariablesArray(lnglN) = Mid$(slLine, 1, lnglB1Pos - 1)
      End If
    Next lnglN
    For lnglN = 0 To UBound(slPVariablesArray)
      slPVariablesArray(lnglN) = Replace(slPVariablesArray(lnglN), "}", "")
    Next lnglN

  End If

  lnglLocalVariablesCount = lnglParameterCount + lnglDimCount
  If lnglLocalVariablesCount > 0 Then
    ReDim slLocalVariablesArray(lnglLocalVariablesCount - 1)
    For lnglN = 0 To lnglParameterCount - 1
      slLocalVariablesArray(lnglN) = slPVariablesArray(lnglN)
    Next lnglN

    For lnglN = 0 To lnglDimCount - 1
      slLocalVariablesArray(lnglN + lnglParameterCount) = slDVariablesArray(lnglN)
    Next lnglN
  End If

  ' ------------------------------------------------------------------
  ' Get REAL "End Sub" line.

  lnglEndProcLine = lnglModuleProcEndLine
  Do
    slLine = vbcmlCodeModule.Lines(lnglEndProcLine, 1)
    If Left$(slLine, 4) = "End " Then
      Exit Do
    End If
```

```
  lnglEndProcLine = lnglEndProcLine - 1
Loop

' ------------------------------------------------------------------
' Put the procedure code into an array.

slCodeLines = vbcmlCodeModule.Lines(lnglModuleProcStartLine, lnglProcCountLines)
slCodeArray = Split(slCodeLines, vbCrLf)

' ------------------------------------------------------------------
' Create a JOINED lines array.

lnglArrayIndex = 0
lnglCLine = lnglModuleProcStartLine

Do

  If lnglCLine > lnglEndProcLine Then
    Exit Do
  End If

  slLine = fncGetSingleLIneFromModule(vbcmlCodeModule, lnglCLine)

  ReDim Preserve slJoinedCodeArray(lnglArrayIndex)
  slJoinedCodeArray(lnglArrayIndex) = slLine
  lnglArrayIndex = lnglArrayIndex + 1

Loop

' ------------------------------------------------------------------
' Collect code "metrics".

' The passed array should be of JOINED lines.
subGetMetrics _
        slJoinedCodeArray(), _
        lnglMaxDepth, _
        lnglBlankLines, _
        lnglCodeLines, _
        lnglCommentLines, _
        lnglTotalOperators, _
        lnglUniqueOperators, _
        lnglTotalOperandsCount, _
        lnglUniqueOperandsCount, _
```

```
        lnglLongLines

' ------------------------------------------------------------------
' Set up the module variables.

lngmEndProcLine = lnglEndProcLine
smCurrentLineText = slCurrentLineText
smSelection = slSelection
lngmFirstDimLine = lnglFirstDimLine
smCodeArray = slCodeArray
smDimsArray = slDimsArray
lngmDimCount = lnglDimCount
lngmSCol = lnglSCol
lngmELine = lnglELine
lngmECol = lnglECol
Set vbpmProject = vbplProject
Set vbcmComponent = vbclComponent
Set vbcmmCodeModule = vbcmlCodeModule
Set vbcpmCodePane = vbcplCodePane
smProjectName = slProjectName
smModuleName = slModuleName
smProcedureName = slProcedureName
lngmCurrentModuleLineNum = lnglCurrentModuleLineNum
lngmModuleProcStartLine = lnglModuleProcStartLine
lngmModuleProcBodyLine = lnglModuleProcBodyLine
lngmProcCountLines = lnglProcCountLines
lngmModuleProcEndLine = lnglModuleProcEndLine
lngmLastDimLinePlus1 = lnglLastDimLinePlus1
lngmLineAfterDefinitionPlus1 = lnglLineAfterDefinitionPlus1
blnmNonConsecutiveDims = blnlNonConsecutiveDims
lngmMaxDepth = lnglMaxDepth
lngmBlankLines = lnglBlankLines
lngmCodeLines = lnglCodeLines
lngmCommentLines = lnglCommentLines
lngmTotalOperators = lnglTotalOperators
lngmUniqueOperators = lnglUniqueOperators
lngmTotalOperandsCount = lnglTotalOperandsCount
lngmUniqueOperandsCount = lnglUniqueOperandsCount
lngmLongLines = lnglLongLines
lngmParameterCount = lnglParameterCount
smParameterArray = slParamsArray
smLocalVariablesArray = slLocalVariablesArray
lngmLocalVariablesCount = lnglLocalVariablesCount
```

```
    smFirstIndent = slfirstindent

    
    ' ************************************************************************
End Sub
Public Property Get SCol() As Long
    SCol = lngmSCol
' ************************************************************************
End Property
Public Property Get ELine() As Long
    ELine = lngmELine
' ************************************************************************
End Property
Public Property Get ECol() As Long
    ECol = lngmECol
' ************************************************************************
End Property
Public Property Get Project() As VBIDE.VBProject
    Set Project = vbpmProject
' ************************************************************************
End Property
Public Property Get Component() As VBIDE.VBComponent
    Set Component = vbcmComponent
' ************************************************************************
End Property
Public Property Get CodeModule() As VBIDE.CodeModule
    Set CodeModule = vbcmmCodeModule
' ************************************************************************
End Property
Public Property Get CodePane() As VBIDE.CodePane
    Set CodePane = vbcpmCodePane
' ************************************************************************
End Property
Public Property Get ProjectName() As String
    ProjectName = smProjectName
' ************************************************************************
End Property
Public Property Get ModuleName() As String
    ModuleName = smModuleName
' ************************************************************************
End Property
Public Property Get ProcedureName() As String
    ProcedureName = smProcedureName
```

```
' ************************************************************************
End Property
Public Property Get CurrentModuleLineNum() As Long
CurrentModuleLineNum = lngmCurrentModuleLineNum
' ************************************************************************
End Property
Public Property Get ModuleProcStartLine() As Long
ModuleProcStartLine = lngmModuleProcStartLine
' ************************************************************************
End Property
Public Property Get ModuleProcBodyLine() As Long
ModuleProcBodyLine = lngmModuleProcBodyLine
' ************************************************************************
End Property
Public Property Get ProcCountLines() As Long
ProcCountLines = lngmProcCountLines
' ************************************************************************
End Property
Public Property Get ModuleProcEndLine() As Long
ModuleProcEndLine = lngmModuleProcEndLine
' ************************************************************************
End Property
Public Property Get CurrentLineText() As String
CurrentLineText = smCurrentLineText
' ************************************************************************
End Property
Public Property Get CodeArray() As Variant
CodeArray = smCodeArray
' ************************************************************************
End Property
Public Property Get Selection() As String
Selection = smSelection
' ************************************************************************
End Property
Public Property Get FirstDimLine() As Long
FirstDimLine = lngmFirstDimLine
' ************************************************************************
End Property
Public Property Get LastDimLinePlus1() As Long
LastDimLinePlus1 = lngmLastDimLinePlus1
' ************************************************************************
End Property
Public Property Get LineAfterDefinitionPlus1() As Long
```

```
    LineAfterDefinitionPlus1 = lngmLineAfterDefinitionPlus1
    ' ********************************************************************
    End Property
    Public Property Get EndProcLine() As Long
    EndProcLine = lngmEndProcLine
    ' ********************************************************************
    End Property
    Public Property Get DimCount() As Long
    DimCount = lngmDimCount
    ' ********************************************************************
    End Property
    Public Property Get DimsArray() As Variant
    DimsArray = smDimsArray
    ' ********************************************************************
    End Property
    Public Property Get MaxDepth() As Long
    MaxDepth = lngmMaxDepth
    ' ********************************************************************
    End Property
    Public Property Get BlankLines() As Long
    BlankLines = lngmBlankLines
    ' ********************************************************************
    End Property
    Public Property Get CodeLines() As Long
    ' ********************************************************************
    CodeLines = lngmCodeLines
    End Property
    Public Property Get CommentLines() As Long
    CommentLines = lngmCommentLines
    ' ********************************************************************
    End Property
    Public Property Get TotalOperators() As Long
    TotalOperators = lngmTotalOperators
    ' ********************************************************************
    End Property
    Public Property Get UniqueOperators() As Long
    UniqueOperators = lngmUniqueOperators
    ' ********************************************************************
    End Property
    Public Property Get TotalOperandsCount() As Long
    TotalOperandsCount = lngmTotalOperandsCount
    ' ********************************************************************
    End Property
```

```
Public Property Get UniqueOperandsCount() As Long
UniqueOperandsCount = lngmUniqueOperandsCount
' ************************************************************************
End Property
Public Property Get LongLines() As Long
LongLines = lngmLongLines
' ************************************************************************
End Property
Public Property Get ParameterArray() As Long
smParameterArray = smParameterArray
' ************************************************************************
End Property
Public Property Get ParameterCount() As Long
ParameterCount = lngmParameterCount
' ************************************************************************
End Property
Public Property Get LocalVariablesArray() As Variant
LocalVariablesArray = smLocalVariablesArray
' ************************************************************************
End Property
Public Property Get LocalVariablesCount() As Long
LocalVariablesCount = lngmLocalVariablesCount
' ************************************************************************
End Property
Public Property Get FirstIndent() As String
FirstIndent = smFirstIndent
' ************************************************************************
End Property

Private Function fncRemoveDoubleSpaces( _
        spLine As String _
        ) _
        As String

If LenB(spLine) = 0 Then
  fncRemoveDoubleSpaces = ""
  Exit Function
End If

Do

  If InStr(spLine, "  ") > 0 Then
    spLine = Replace(spLine, "  ", " ")
```

```
  Else
    Exit Do
  End If

Loop

fncRemoveDoubleSpaces = spLine
' ****************************************************************************
End Function
Private Sub subGetMetrics( _
        spCodeArray() As String, _
        lngpMaxDepth As Long, _
        lngpBlankLines As Long, _
        lngpCodeLines As Long, _
        lngpCommentLines As Long, _
        lngpTotalOperators As Long, _
        lngpUniqueOperators As Long, _
        lngpTotalOperandsCount As Long, _
        lngpUniqueOperandsCount As Long, _
        lngpLongLines As Long _
        )
' I'm defining an OPERAND as any single item that is
'  on either side of an OPERATOR that appears on the right
'  side of an equation.
'
'     A = B + C - D * 20
'
'     Operands are B, C, D, 20
'
' I'm also delibearately avoiding anything with
'  multiple equates.
'
'      A = Iff(B = C+D = 20))
'
' ### The passed code array should be an
'      array of JOINED lines.
'

Dim blnlFound As Boolean
Dim lnglAfterLen As Long
Dim lnglBeforeLen As Long
Dim lnglBlankLines As Long
Dim lnglBranchCount As Long
```

	Dim lnglBranchCountsArray() As Long
	Dim lnglBranchlen As Long
	Dim lnglCLine As Long
	Dim lnglCodeLines As Long
	Dim lnglCommentLines As Long
	Dim lnglDeleteItemLen As Long
	Dim lnglEndProcLine As Long
	Dim lnglEqPos As Long
	Dim lnglLongLines As Long
	Dim lnglM As Long
	Dim lnglMaxDepth As Long
	Dim lnglN As Long
	Dim lnglNestDepth As Long
	Dim lnglNumBranches As Long
	Dim lnglNumOperators As Long
	Dim lnglOperandIndex As Long
	Dim lnglOperatorCount As Long
	Dim lnglOperatorCountsArray() As Long
	Dim lnglOperatorlen As Long
	Dim lnglTotalBranches As Long
	Dim lnglTotalOperandsCount As Long
	Dim lnglTotalOperators As Long
	Dim lnglTrimmedLineLen As Long
	Dim lnglUniqueBranches As Long
	Dim lnglUniqueOperandsCount As Long
	Dim lnglUniqueOperators As Long
	Dim slAllOperansdArray() As String
	Dim slBranch As String
	Dim slBranchArray() As String
	Dim slBranchEnd As String
	Dim slBranchEndArray() As String
	Dim slBranchEnds As String
	Dim slBranches As String
	Dim slCn As String
	Dim slCodeLinesA As String
	Dim slCodeLinesAArray() As String
	Dim slCodeLinesBArray() As String
	Dim slDeleteItem As String
	Dim slDeleteLines As String
	Dim slDeleteLinesArray() As String
	Dim slDQ As String
	Dim slLineArray() As String
	Dim slLineItem As String

```
    Dim slOperandLine As String
    Dim slOperator As String
    Dim slOperatorArray() As String
    Dim slOperators As String
    Dim slTrimmedLine As String
    Dim slUniqueOperandsArray() As String

    slDQ = Chr(34)
    slCn = " _"

    slOperators = "=,>,<,>=,<=,<>,And,Or,Xor,Not,+,-,*,/,\,^,Mod,&"
    slOperatorArray = Split(slOperators, ",")

    slBranches = "If,ElseIf,Do,Select Case,Case,For"
    slBranchEnds = "End,Loop,Next"
    slBranchArray = Split(slBranches, ",")
    slBranchEndArray = Split(slBranchEnds, ",")

    slDeleteLines = "Dim ,Const ,Static ,Sub ,End Sub ,End Sub,Function ,End Function,Private ,Public ,Exit Do"
    slDeleteLinesArray = Split(slDeleteLines, ",")

    ' The array should be JOINED!
    'slCodeLinesAArray = spCodeArray

    slCodeLinesAArray = fncGetJoinedArrayFromArray(spCodeArray)

    lnglEndProcLine = UBound(slCodeLinesAArray)
    ReDim slCodeLinesBArray(0)

    lnglOperandIndex = 0
    ReDim slAllOperansdArray(lnglOperandIndex)

    ' Remove and count comments and blank lines.
    ' Count code lines and long lines.
    ' Get all operands.
    lnglCLine = 0
    Do

      slTrimmedLine = Trim$( _
                  slCodeLinesAArray(lnglCLine))
      lnglTrimmedLineLen = Len(slTrimmedLine)

      If lnglTrimmedLineLen <> 0 Then
```

```
    ' Comment?
    If Left$(slTrimmedLine, 1) = "'" Then
      slCodeLinesAArray(lnglCLine) = ""
      lnglCommentLines = lnglCommentLines + 1
      slTrimmedLine = ""
    ElseIf InStr(2, slTrimmedLine, "'") > 0 Then
      lnglCommentLines = lnglCommentLines + 1
      slTrimmedLine = Mid$(slTrimmedLine, 1, InStr(2, slTrimmedLine, "'") - 1)
    End If

    ' Zap this line?
    lnglTrimmedLineLen = Len(slTrimmedLine)
    If lnglTrimmedLineLen > 0 Then
      For lnglN = 0 To UBound(slDeleteLinesArray)
        slDeleteItem = slDeleteLinesArray(lnglN)
        lnglDeleteItemLen = Len(slDeleteItem)
        If Left$(slTrimmedLine, lnglDeleteItemLen) = slDeleteItem Then
          slTrimmedLine = ""
          Exit For
        End If
      Next lnglN
    End If

    ' Code Line?
    lnglTrimmedLineLen = Len(slTrimmedLine)
    If lnglTrimmedLineLen > 0 Then

      lnglCodeLines = lnglCodeLines + 1
      slCodeLinesBArray(UBound(slCodeLinesBArray)) = slTrimmedLine
      ReDim Preserve slCodeLinesBArray(UBound(slCodeLinesBArray) + 1)

      ' Long Line?
      If lnglTrimmedLineLen > 60 Then
        lnglLongLines = lnglLongLines + 1
      End If

      ' Put the TRIMMED code line back.
      slCodeLinesAArray(lnglCLine) = slTrimmedLine

      ' Check Nesting Depth.
      slLineArray = Split(slTrimmedLine, " ")
      For lnglN = 0 To UBound(slBranchArray)
```

```
        If slLineArray(0) = slBranchArray(lnglN) Then
          lnglNestDepth = lnglNestDepth + 1
          Exit For
        End If
      Next lnglN

      If lnglNestDepth > lnglMaxDepth Then
        lnglMaxDepth = lnglNestDepth
      End If

      ' Check UnNesting Depth.
      For lnglN = 0 To UBound(slBranchEndArray)
        If slLineArray(0) = slBranchEndArray(lnglN) Then
          lnglNestDepth = lnglNestDepth - 1
          Exit For
        End If
      Next lnglN


      ' Get OPERANDS as defined above.
      ' Assume ONE =.
      lnglEqPos = InStr(slTrimmedLine, "=")
      If lnglEqPos > 0 Then
        slOperandLine = Mid$(slTrimmedLine, lnglEqPos + 2)

        ' Pre process line.
        ' Try and get just operators and/or operands on the line.
        '  This is easier in VBA than C/C#/C++/Java/PHP etc and also because
        '  of our descision about what an operand is.
        slOperandLine = Replace(slOperandLine, "(", " ")
        slOperandLine = Replace(slOperandLine, ")", " ")
        slOperandLine = Replace(slOperandLine, ",", " ")
        slOperandLine = Replace(slOperandLine, "=", " ")
        slOperandLine = fncRemoveDoubleSpaces(slOperandLine)

        slLineArray = Split(slOperandLine, " ")

        For lnglN = 0 To UBound(slLineArray)
          slLineItem = slLineArray(lnglN)

          For lnglM = 0 To UBound(slOperatorArray)

            If slLineItem = slOperatorArray(lnglM) Then
```

```
            ' If it's an operator grab the previous item.
            slAllOperansdArray(lnglOperandIndex) = slLineArray(lnglN - 1)

            lnglOperandIndex = lnglOperandIndex + 1
            ReDim Preserve slAllOperansdArray(lnglOperandIndex)

            ' Do we need to catch the last "item"?
            If lnglN = UBound(slLineArray) - 1 Then

              slAllOperansdArray(lnglOperandIndex) = slLineArray(UBound(slLineArray))

              lnglOperandIndex = lnglOperandIndex + 1
              ReDim Preserve slAllOperansdArray(lnglOperandIndex)

            End If

            Exit For

          End If

        Next lnglM

      Next lnglN

    End If

  End If

 Else
  lnglBlankLines = lnglBlankLines + 1
 End If

 lnglCLine = lnglCLine + 1
 If lnglCLine > lnglEndProcLine Then
  Exit Do
 End If

Loop

slCodeLinesA = Join(slCodeLinesBArray, " ")
slCodeLinesA = Replace(slCodeLinesA, vbCrLf, " ")
slCodeLinesA = Replace(slCodeLinesA, "(", " ( ")
```

```
slCodeLinesA = Replace(slCodeLinesA, ")", " ) ")
slCodeLinesA = Replace(slCodeLinesA, ",", " ")
slCodeLinesA = fncRemoveDoubleSpaces(slCodeLinesA)

lnglNumOperators = UBound(slOperatorArray)
ReDim lnglOperatorCountsArray(lnglNumOperators)

' Count the operators.
For lnglN = 0 To lnglNumOperators

  slOperator = slOperatorArray(lnglN)
  lnglOperatorlen = Len(slOperator)

  lnglBeforeLen = Len(slCodeLinesA)
  slCodeLinesA = Replace(slCodeLinesA, " " & slOperator & " ", "  ")

  lnglAfterLen = Len(slCodeLinesA)
  lnglOperatorCount = (lnglBeforeLen - lnglAfterLen) / lnglOperatorlen
  If lnglOperatorCount > 0 Then
    lnglUniqueOperators = lnglUniqueOperators + 1
  End If

  lnglOperatorCountsArray(lnglN) = lnglOperatorCount
  lnglTotalOperators = lnglTotalOperators + lnglOperatorCount

Next lnglN

slCodeLinesA = fncRemoveDoubleSpaces(slCodeLinesA)

lnglNumBranches = UBound(slBranchArray)
ReDim lnglBranchCountsArray(lnglNumBranches)

' Count the branches.
For lnglN = 0 To UBound(slBranchEndArray)
  slBranchEnd = slBranchEndArray(lnglN)
  slCodeLinesA = Replace(slCodeLinesA, " " & slBranchEnd & " ", "  ")
Next lnglN

For lnglN = 0 To lnglNumBranches

  slBranch = slBranchArray(lnglN)
  lnglBranchlen = Len(slBranch)
```

```
    lnglBeforeLen = Len(slCodeLinesA)
    slCodeLinesA = Replace(slCodeLinesA, " " & slBranch & " ", "  ")
    lnglAfterLen = Len(slCodeLinesA)

    lnglBranchCount = (lnglBeforeLen - lnglAfterLen) / lnglBranchlen
    If lnglBranchCount > 0 Then
      lnglUniqueBranches = lnglUniqueBranches + 1
    End If

    lnglBranchCountsArray(lnglN) = lnglBranchCount
    lnglTotalBranches = lnglTotalBranches + lnglBranchCount

  Next lnglN

  ' Cull the Operands for Unique.
  ReDim slUniqueOperandsArray(0)
  For lnglN = 0 To UBound(slAllOperansdArray)
    blnlFound = False
    For lnglM = 0 To UBound(slUniqueOperandsArray)
      If slAllOperansdArray(lnglN) = slUniqueOperandsArray(lnglM) Then
        ' Got it already.
        blnlFound = True
        Exit For
      End If
    Next lnglM

    If blnlFound = False Then
      slUniqueOperandsArray(UBound(slUniqueOperandsArray)) = slAllOperansdArray(lnglN)
      ReDim Preserve slUniqueOperandsArray(UBound(slUniqueOperandsArray) + 1)
    End If

  Next lnglN

  lnglTotalOperandsCount = UBound(slAllOperansdArray)
  lnglUniqueOperandsCount = UBound(slUniqueOperandsArray)

  lngpMaxDepth = lnglMaxDepth
  lngpBlankLines = lnglBlankLines
  lngpCodeLines = lnglCodeLines
  lngpCommentLines = lnglCommentLines
  lngpTotalOperators = lnglTotalOperators
  lngpUniqueOperators = lnglUniqueOperators
  lngpTotalOperandsCount = lnglTotalOperandsCount
```

```
    lngpUniqueOperandsCount = lnglUniqueOperandsCount
    lngpLongLines = lnglLongLines

    ' ********************************************************************
    End Sub
    Private Function fncGetJoinedArrayFromArray( _
                    spLines() As String _
                    )
    ' Return an array of "joined" lines.
    '
    ' I built this mostly to return joined
    '  declaration lines though it will
    '  work for any set of passed lines.
    '

    Dim lnglN As Long
    Dim lnglLineIndex As Long
    Dim lnglNumJoinedLines As Long
    Dim slJoinedLine As String
    Dim slJoinedLinesArray() As String
    Dim slCommentTest As String
    Dim lnglUB As Long
    Dim blnlComment As Boolean
    Dim slTrimmedLine As String
    Dim slJoinChr As String


    slJoinChr = " _"
    lnglLineIndex = 0
    lnglNumJoinedLines = 0
    lnglUB = UBound(spLines)

    ' First COUNT the lines for the array redim.
    ' Redim preserve Array(UBound(Array)+1) will work inside the
    '  loop but there seems to be quite an overhead for that.
    Do

      slTrimmedLine = Trim$(spLines(lnglLineIndex))

      If Left$(slTrimmedLine, 1) = "'" Then
        blnlComment = True
      End If
```

```
    If blnlComment = True Then

      If LenB(slTrimmedLine) > 2 Then
        If Right$(slTrimmedLine, 2) <> slJoinChr Then

          ' End of continuations in comments.
          blnlComment = False
          lnglLineIndex = lnglLineIndex + 1

        Else

          ' Comment and continuation.
          lnglLineIndex = lnglLineIndex + 1

        End If

      Else

        ' End of continuations in comments.
        blnlComment = False
        lnglLineIndex = lnglLineIndex + 1

      End If

    Else

      ' Not in Comments.


      ' lnglLineIndex is incremented in the function.
      slJoinedLine = _
          fncGetSingleLineFromArray(spLines(), lnglLineIndex)

    End If

    lnglNumJoinedLines = lnglNumJoinedLines + 1

    If lnglLineIndex >= lnglUB Then
      Exit Do
    End If

  Loop

```

```
' Redim the EXACT array size.
ReDim slJoinedLinesArray(lnglNumJoinedLines)

' Populate our array.
lnglLineIndex = 0
lnglNumJoinedLines = 0

Do

  slTrimmedLine = Trim$(spLines(lnglLineIndex))

  If Left$(slTrimmedLine, 1) = "'" Then
    blnlComment = True
  End If

  If blnlComment = True Then

    If LenB(slTrimmedLine) > 2 Then
      If Right$(slTrimmedLine, 2) <> slJoinChr Then

        ' End of continuations in comments.
        slJoinedLine = spLines(lnglLineIndex)
        blnlComment = False
        lnglLineIndex = lnglLineIndex + 1

      Else

        ' Comment and continuation.
        slJoinedLine = spLines(lnglLineIndex)
        lnglLineIndex = lnglLineIndex + 1

      End If

    Else

      ' End of continuations in comments.
      slJoinedLine = spLines(lnglLineIndex)
      blnlComment = False
      lnglLineIndex = lnglLineIndex + 1

    End If

  Else
```

```
    ' Not in Comments.

     ' lnglLineIndex is incremented in the function.
     slJoinedLine = _
         fncGetSingleLineFromArray(spLines(), lnglLineIndex)

   End If

   slJoinedLinesArray(lnglNumJoinedLines) = slJoinedLine
   lnglNumJoinedLines = lnglNumJoinedLines + 1

   If lnglLineIndex >= lnglUB Then
     Exit Do
   End If

 Loop

 fncGetJoinedArrayFromArray = slJoinedLinesArray()
 ' ********************************************************************
 End Function
 Private Function fncGetSingleLineFromArray( _
         spCodeArray() As String, _
         lngpArrayIndex As Long)
 ' Join lines with continuation character up into
 '  a single line.
 ' The source is an array rather than an actual
 '  module.
 ' Increment the given line number.
 '
 ' Note that it IS possible to have lines with 1 or 2 chrs
 '  on them.
 '
 ' Assumes NO line numbers.
 '
 ' Possible scenarios...
 '   Blank line*
 '   Comment Line*
 '   Code line 0 with no continuation*
 '   Code line > 0 with no continuation and no continuation on prev line*
 '   Code line with continuation and no continuation on prev line
 '   Code line with continuation and continuation on prev line
```

```
'

Dim blnlCommentLine As Boolean
Dim blnlLoopDown As Boolean
Dim blnlLoopUp As Boolean
Dim lnglArrayIndex As Long
Dim lnglEndLine As Long
Dim lnglMaxLines As Long
Dim slGetLine As String
Dim slJoinChr As String
Dim slNewLine As String
Dim slOriginalLine As String
Dim slTrimmedLine As String
Dim slCommentLine As String

slJoinChr = " _"
slNewLine = ""
blnlLoopDown = False
blnlLoopUp = False
lnglEndLine = lngpArrayIndex
lnglArrayIndex = lngpArrayIndex
slOriginalLine = spCodeArray(lnglArrayIndex)
slTrimmedLine = Trim$(slOriginalLine)
lnglMaxLines = UBound(spCodeArray)

If LenB(slTrimmedLine) = 0 Then

  ' Blank line.
  ' No need to go through the whole process.
  lngpArrayIndex = lnglArrayIndex + 1
  fncGetSingleLineFromArray = slOriginalLine
  Exit Function

End If

' Is this line 0 with no continuation.
If lnglArrayIndex = 0 Then
  If LenB(slTrimmedLine) > 2 Then
    If Right$(slTrimmedLine, 2) <> slJoinChr Then

      ' No continuation and no previous line.
      lngpArrayIndex = lnglArrayIndex + 1
      fncGetSingleLineFromArray = slOriginalLine
```

```
      Exit Function

    End If

  Else

    ' No continuation and no previous line.
    lngpArrayIndex = lnglArrayIndex + 1
    fncGetSingleLineFromArray = slOriginalLine
    Exit Function

  End If
End If

' Is this a line with no continuation and no continuation on
'  previous line.
If LenB(slTrimmedLine) > 2 Then
  If Right$(slTrimmedLine, 2) <> slJoinChr Then

    slGetLine = Trim$(spCodeArray(lnglArrayIndex - 1))
    If LenB(slGetLine) > 2 Then
      If Right$(slGetLine, 2) <> slJoinChr Then

        lngpArrayIndex = lnglArrayIndex + 1
        fncGetSingleLineFromArray = slOriginalLine
        Exit Function

      End If

    Else

        lngpArrayIndex = lnglArrayIndex + 1
        fncGetSingleLineFromArray = slOriginalLine
        Exit Function

    End If
  End If
Else

  slGetLine = Trim$(spCodeArray(lnglArrayIndex - 1))
  If LenB(slGetLine) > 2 Then
    If Right$(slGetLine, 2) <> slJoinChr Then

```

```
      lngpArrayIndex = lnglArrayIndex + 1
      fncGetSingleLineFromArray = slGetLine
      Exit Function

    End If
  End If

End If

' If we get here the line must be at least contain the original one.
slNewLine = slTrimmedLine

' We should now be at the top of any continuation lines if
'  there are any.
slGetLine = Trim$(spCodeArray(lnglArrayIndex))

' Does this line have a continuation?
If LenB(slTrimmedLine) > 2 Then
  If Right$(slTrimmedLine, 2) = slJoinChr Then

    ' There is a continuation on the original line.
    ' Loop Down.
    blnlLoopDown = True
    Do
      lnglArrayIndex = lnglArrayIndex + 1
      If lnglArrayIndex > lnglMaxLines Then
        Exit Do
      End If

      slGetLine = Trim$(spCodeArray(lnglArrayIndex))
      slNewLine = slNewLine & slGetLine

      If Right$(slNewLine, 2) <> slJoinChr Then

        lnglEndLine = lnglArrayIndex
        Exit Do

      End If

    Loop

  End If
End If
```

```
' Go back to the original line and check upwards.
' We've checked for the first line and no continuation
'  chr so there must be lines above this.
lnglArrayIndex = lngpArrayIndex
Do
  lnglArrayIndex = lnglArrayIndex - 1
  If lnglArrayIndex < 0 Then
    Exit Do
  End If

  slGetLine = Trim$(spCodeArray(lnglArrayIndex))

  If LenB(slGetLine) > 2 Then
    If Right$(slGetLine, 2) <> slJoinChr Then

      ' No continuation on previous line.
      Exit Do

    Else

      ' Continuation chr on previous line.
      slNewLine = slGetLine & slNewLine

    End If

  Else

    ' No continuation chr.
    Exit Do

  End If

Loop

' Get rid of the continuation chrs.
slNewLine = Replace(slNewLine, slJoinChr, " ")

' Set up any &s correctly.
slNewLine = Replace(slNewLine, "&", " & ")

' Get rid of lots of spaces.
slNewLine = fncRemoveDoubleSpaces(slNewLine)
```

```
' Increment original line number.
lngpArrayIndex = lnglEndLine + 1

fncGetSingleLineFromArray = slNewLine
' ************************************************************************
End Function
Private Function fncGetSingleLIneFromModule( _
      vbcmpCodeModule As VBIDE.CodeModule, _
      lngpCurrentModuleLineNum As Long _
      )
' Join lines with continuation character up into
'  a single line.
' Get the lines from the actual module.
' Increment the given line number.
'

Dim ilCommentPos As Integer
Dim lnglCurrentLine As Long
Dim slJoinChr As String
Dim slGetLine As String
Dim slOldLine As String
Dim slNewLine As String
Dim blnlLoopUp As Boolean
Dim blnlLoopDown As Boolean

slJoinChr = " _"

slOldLine = Trim$(vbcmpCodeModule.Lines(lngpCurrentModuleLineNum, 1))
lnglCurrentLine = lngpCurrentModuleLineNum

blnlLoopDown = False
blnlLoopUp = False

If LenB(slOldLine) > 2 Then
  If Right$(slOldLine, 2) = slJoinChr Then
    blnlLoopDown = True
  End If
Else
  fncGetSingleLIneFromModule = slOldLine
  lngpCurrentModuleLineNum = lngpCurrentModuleLineNum + 1

  Exit Function
```

```
    End If

    slGetLine = Trim$(vbcmpCodeModule.Lines(lnglCurrentLine - 1, 1))

    If LenB(slGetLine) > 2 Then
      If Right$(slGetLine, 2) = slJoinChr Then
        slNewLine = slOldLine & " " & slGetLine
        blnlLoopUp = True
      End If
    Else
      fncGetSingleLIneFromModule = slOldLine
      lngpCurrentModuleLineNum = lngpCurrentModuleLineNum + 1
      Exit Function
    End If

    If blnlLoopDown = True Then

      Do
        lnglCurrentLine = lnglCurrentLine + 1
        slGetLine = Trim$(vbcmpCodeModule.Lines(lnglCurrentLine, 1))
        slNewLine = slNewLine & slGetLine

        If Right$(slNewLine, 2) <> slJoinChr Then
          Exit Do
        End If

      Loop

    End If

    If blnlLoopUp = True Then

      ' We already have the previous line.
      lnglCurrentLine = lngpCurrentModuleLineNum - 1
      Do
        lnglCurrentLine = lnglCurrentLine - 1
        slGetLine = vbcmpCodeModule.Lines(lnglCurrentLine, 1)
        If Right$(slGetLine, 2) <> slJoinChr Then
          Exit Do
        End If
        slNewLine = slGetLine & slNewLine

      Loop
```

```
                
                End If
                
                
                ' Get rid of the continuation chrs.
                slNewLine = Replace(slNewLine, slJoinChr, "")
                
                ' Set up any &s correctly.
                slNewLine = Replace(slNewLine, "&", " & ")
                
                ' Get rid of lots of spaces.
                slNewLine = fncRemoveDoubleSpaces(slNewLine)
                
                lngpCurrentModuleLineNum = lngpCurrentModuleLineNum + 1
                
                fncGetSingleLIneFromModule = slNewLine
                ' ****************************************************************
                End Function
```