--- Table of Contents: ---

--- ThisWorkbook ---

...

--- README - Sheet22 ---

...

```
Option Explicit
```

--- INI ---

```
Option Explicit
#If VBA7 Then
Private Declare PtrSafe Function GetPrivateProfileString Lib "kernel32" Alias  _
"GetPrivateProfileStringA" (ByVal lpApplicationName As String, ByVal lpKeyName As Any, ByVal  _
lpDefault As String, ByVal lpReturnedString As String, ByVal nSize As Long, ByVal lpFileName As  _
String) As Long
Private Declare PtrSafe Function WritePrivateProfileString Lib "kernel32" Alias  _
"WritePrivateProfileStringA" (ByVal lpApplicationName As String, ByVal lpKeyName As Any, ByVal  _
lpString As Any, ByVal lpFileName As String) As Long
Private Declare PtrSafe Function GetPrivateProfileSection Lib "kernel32" Alias  _
"GetPrivateProfileSectionA" (ByVal lpAppName As String, ByVal lpReturnedString As String, ByVal  _
nSize As Long, ByVal lpFileName As String) As Long
#Else
Private Declare Function GetPrivateProfileString Lib "kernel32" Alias "GetPrivateProfileStringA" ( _        INI.
ByVal lpApplicationName As String, ByVal lpKeyName As Any, ByVal lpDefault As String, ByVal  _
lpReturnedString As String, ByVal nSize As Long, ByVal lpFileName As String) As Long
Private Declare Function WritePrivateProfileString Lib "kernel32" Alias  _
"WritePrivateProfileStringA" (ByVal lpApplicationName As String, ByVal lpKeyName As Any, ByVal  _
lpString As Any, ByVal lpFileName As String) As Long
Private Declare Function GetPrivateProfileSection Lib "kernel32" Alias "GetPrivateProfileSectionA" ( _      INI.
ByVal lpAppName As String, ByVal lpReturnedString As String, ByVal nSize As Long, ByVal lpFileName  _
As String) As Long
#End If
Public Function IniSections(IniFile As String) As Variant
    IniSections = Split(Replace(Replace(Join(Filter(Split(Replace(TxtRead(IniFile), vbLf, vbNewLine) _
    , vbNewLine), "[", True), vbNewLine), "[", ""), "]", ""), vbNewLine)
End Function
Public Function IniReadSection(filename As String, Section As String) As Variant
    Dim RetVal      As String * 255
    Dim v           As Long: v = GetPrivateProfileSection(Section, RetVal, 255, filename)
    Dim S           As String: S = Left(RetVal, v + 0)
    Dim VL          As Variant: VL = Split(S, Chr$(0))
    If UBound(VL) = -1 Then IniReadSection = "": Exit Function
    VL = ArrayRemoveEmptyElements(VL)
    IniReadSection = VL
End Function
Public Function IniSectionKeys(filename As String, Section As String) As Variant
    Dim arr         As Variant
    arr = IniReadSection(filename, Section)
    Dim out         As Variant
    ReDim out(UBound(arr))
    Dim i           As Long
    For i = LBound(arr) To UBound(arr)
        out(i) = Trim(Split(arr(i), "=")(0))
    Next i
    IniSectionKeys = out
End Function
Public Function IniReadKey(IniFileName As String, ByVal Sect As String, ByVal Keyname As String) As  _
String
    Dim Worked      As Long
    Dim RetStr      As String * 128
    Dim StrSize     As Long
    Dim iNoOfCharInIni As Long: iNoOfCharInIni = 0
    Dim sIniString  As String: sIniString = ""
    If Sect = "" Or Keyname = "" Then
        MsgBox "Section Or Key To Read Not Specified !!!", vbExclamation, "INI"
    Else
        Dim sProfileString As String: sProfileString = ""
        RetStr = Space(128)
        StrSize = Len(RetStr)
        Worked = GetPrivateProfileString(Sect, Keyname, "", RetStr, StrSize, IniFileName)
        If Worked Then
            iNoOfCharInIni = Worked
            sIniString = Left$(RetStr, Worked)
        End If
    End If
    IniReadKey = sIniString
End Function
Public Sub IniWrite(IniFileName As String, ByVal Sect As String, ByVal Keyname As String, ByVal  _
Wstr As String)
```

```
    Dim Worked      As Long
    Dim iNoOfCharInIni As Long
    iNoOfCharInIni = 0
    Dim sIniString  As String: sIniString = ""
    If Sect = "" Or Keyname = "" Then
        MsgBox "Section Or Key To Write Not Specified !!!", vbExclamation, "INI"
    Else
        Worked = WritePrivateProfileString(Sect, Keyname, Wstr, IniFileName)
        If Worked Then
            iNoOfCharInIni = Worked
            sIniString = Wstr
        End If
    End If
End Sub
Public Function IniSectionExists(IniFile As String, Section As String) As Boolean                         ...
    IniSectionExists = InStr(1, TxtRead(IniFile), "[" & Section & "]") > 0
End Function
Public Function IniKeyExists(IniFile As String, Section As String, key As String) As Boolean              ...
    IniKeyExists = (IniReadKey(IniFile, Section, key) <> "")
End Function
Public Sub TestReadKey()                                                                                    ...
    Debug.Print "INI File: " & ThisWorkbook.Path & "\MyIniFile.ini" & vbCrLf & _
            "Section: SETTINGS" & vbCrLf & _
            "Section Exist: " & IniSectionExists(ThisWorkbook.Path & "\MyIniFile.ini", "SETTINGS")  _
            & vbCrLf & _
            "Key: License" & vbCrLf & _
            "Key Exist: " & IniKeyExists(ThisWorkbook.Path & "\MyIniFile.ini", "SETTINGS",  _
            "License") & vbCrLf & _
            "Key Value: " & Ini_ReadKeyVal(ThisWorkbook.Path & "\MyIniFile.ini", "SETTINGS",  _
            "License")
End Sub
Public Sub TestWriteKey()                                                                                   ...
    If Ini_WriteKeyVal(ThisWorkbook.Path & "\MyIniFile.ini", "SETTINGS", "License",  _
    "JBXR-HHTY-LKIP-HJNB-GGGT") = True Then
        MsgBox "The key was written"
    Else
        MsgBox "An error occurred!"
    End If
End Sub
Public Function Ini_ReadKeyVal(ByVal sIniFile As String, _                                                  ...
        ByVal sSection As String, _
        ByVal sKey As String) As String
    On Error GoTo Error_Handler
    Dim bSectionExists As Boolean
    Dim bKeyExists  As Boolean
    Dim sIniFileContent As String
    Dim aIniLines() As String
    Dim sLine       As String
    Dim i           As Long
    sIniFileContent = ""
    bSectionExists = False
    bKeyExists = False
    If FileExists(sIniFile) = False Then
        MsgBox "The specified ini file: " & vbCrLf & vbCrLf & _
                sIniFile & vbCrLf & vbCrLf & _
                "could not be found.", vbCritical + vbOKOnly, "File not found"
        GoTo Error_Handler_Exit
    End If
    sIniFileContent = TxtRead(sIniFile)
    aIniLines = Split(sIniFileContent, vbLf)
    For i = 0 To UBound(aIniLines)
        sLine = Trim(aIniLines(i))
        sLine = VBA.Replace(sLine, vbTab, vbNullString)
        If InStr(1, sLine, "=") > 0 Then sLine = Join(ArrayTrim(Split(sLine, "=")),
        If bSectionExists = True And Left(sLine, 1) = "[" And Right(sLine, 1) = "]" Then
        End If
        If sLine = "[" & sSection & "]" Then
            bSectionExists = True
        End If
        If bSectionExists = True Then
            If sLine Like sKey & "=*" Then
                bKeyExists = True
                Ini_ReadKeyVal = Mid(sLine, InStr(sLine, "=") + 1)
            End If
        End If
    Next i
Error_Handler_Exit:
    On Error Resume Next
    Exit Function
Error_Handler:
    MsgBox "The following error has occurred" & vbCrLf & vbCrLf & _
            "Error Number: " & Err.Number & vbCrLf & _
            "Error Source: Ini_ReadKeyVal" & vbCrLf & _
            "Error Description: " & Err.Description & _
            Switch(Erl = 0, "", Erl <> 0, vbCrLf & "Line No: " & Erl) _
            , vbOKOnly + vbCritical, "An Error has Occurred!"
    Resume Error_Handler_Exit
End Function
Public Function Ini_WriteKeyVal(ByVal sIniFile As String, _                                    ...
        ByVal sSection As String, _
        ByVal sKey As String, _
        ByVal sValue As String) As Boolean
    On Error GoTo Error_Handler
    Dim bSectionExists As Boolean
    Dim bKeyExists  As Boolean
    Dim sIniFileContent As String
    Dim aIniLines() As String
    Dim sLine       As String
    Dim sNewLine    As String
    Dim i           As Long
    Dim bFileExist  As Boolean
    Dim bInSection  As Boolean
    Dim bKeyAdded   As Boolean
    sIniFileContent = ""
    bSectionExists = False
    bKeyExists = False
    If FileExists(sIniFile) = False Then
        GoTo SectionDoesNotExist
    End If
    bFileExist = True
    sIniFileContent = TxtRead(sIniFile)
    aIniLines = Split(sIniFileContent, vbLf)
    sIniFileContent = ""
    For i = 0 To UBound(aIniLines)
        sNewLine = ""
        sLine = Trim(aIniLines(i))
        If sLine = "[" & sSection & "]" Then
            bSectionExists = True
            bInSection = True
        End If
        If bInSection = True Then
            If sLine <> "[" & sSection & "]" Then
                If Left(sLine, 1) = "[" And Right(sLine, 1) = "]" Then
                    sNewLine = sKey & "=" & sValue
                    i = i - 1
                    bI
                    bKeyAdded = True
                End If
            End If
            If Len(sLine) > Len(sKey) Then
                If Left(sLine, Len(sKey) + 1) = sKey & "=" Then
                    sNewLine = sKey & "=" & sValue
                    bKeyExists = True
                    bKeyAdded = True
                End If
            End If
        End If
        If Len(sIniFileContent) > 0 Then sIniFileContent = sIniFileContent & vbCrLf
        If sNewLine = "" Then
            sIniFileContent = sIniFileContent & sLine
        Else
            sIniFileContent = sIniFileContent & sNewLine
        End If
    Next i
```

```
SectionDoesNotExist:
    If bSectionExists = False Then
        If Len(sIniFileContent) > 0 Then sIniFileContent = sIniFileContent & vbCrLf
        sIniFileContent = sIniFileContent & "[" & sSection & "]"
    End If
    If bKeyAdded = False Then
        sIniFileContent = sIniFileContent & vbCrLf & sKey & "=" & sValue
    End If
    Call TxtOverwrite(sIniFile, sIniFileContent)
    Ini_WriteKeyVal = True
Error_Handler_Exit:
    On Error Resume Next
    Exit Function
Error_Handler:
    MsgBox "The following error has occurred" & vbCrLf & vbCrLf & _
            "Error Number: " & Err.Number & vbCrLf & _
            "Error Source: Ini_WriteKeyVal" & vbCrLf & _
            "Error Description: " & Err.Description & _
            Switch(Erl = 0, "", Erl <> 0, vbCrLf & "Line No: " & Erl) _
            , vbOKOnly + vbCritical, "An Error has Occurred!"
    Resume Error_Handler_Exit
End Function
```

--- zOther ---

```
Option Explicit
Function ArrayColumn(arr As Variant, Col As Long) As Variant
    ArrayColumn = WorksheetFunction.index(arr, 0, Col)
End Function
Function ArrayFilter2d(ByVal sArraY, ByVal ColIndex As Long, ByVal FindStr As String, ByVal  _
HasTitle As Boolean)
    Dim tmpArr, i As Long, j As Long, arr, dic, tmpStr, tmp, Chk As Boolean, TmpVal As Double
    On Error Resume Next
    Set dic = CreateObject("Scripting.Dictionary")
    tmpArr = sArraY
    ColIndex = ColIndex + LBound(tmpArr, 2) - 1
    Chk = (InStr("><=", Left(FindStr, 1)) > 0)
    For i = LBound(tmpArr, 1) - HasTitle To UBound(tmpArr, 1)
        If Chk Then
            TmpVal = CDbl(tmpArr(i, ColIndex))
            If Evaluate(TmpVal & FindStr) Then dic.Add i, ""
        Else
            If UCase(tmpArr(i, ColIndex)) Like UCase(FindStr) Then dic.Add i, ""
        End If
    Next
    If dic.Count > 0 Then
        tmp = dic.Keys
        ReDim arr(LBound(tmpArr, 1) To UBound(tmp) + LBound(tmpArr, 1) - HasTitle, LBound(tmpArr, 2) _
        To UBound(tmpArr, 2))
        For i = LBound(tmpArr, 1) - HasTitle To UBound(tmp) + LBound(tmpArr, 1) - HasTitle
            For j = LBound(tmpArr, 2) To UBound(tmpArr, 2)
                arr(i, j) = tmpArr(tmp(i - LBound(tmpArr, 1) + HasTitle), j)
            Next
        Next
        If HasTitle Then
            For j = LBound(tmpArr, 2) To UBound(tmpArr, 2)
                arr(LBound(tmpArr, 1), j) = tmpArr(LBound(tmpArr, 1), j)
            Next
        End If
    End If
    ArrayFilter2d = arr
End Function
Function GetInputRange(ByRef rInput As Excel.Range, _
        sPrompt As String, _
        sTitle As String, _
        Optional ByVal sDefault As String, _
        Optional ByVal bActivate As Boolean, _
        Optional x, _
        Optional y) As Boolean
    Dim bGotRng     As Boolean
    Dim bEvents     As Boolean
    Dim nAttempt    As Long
    Dim sAddr       As String
    Dim vReturn
    On Error Resume Next
    If Len(sDefault) = 0 Then
        If TypeName(Application.Selection) = "Range" Then
            sDefault = "=" & Application.Selection.Address
            If Len(sDefault) > 240 Then
                sDefault = "=" & Application.ActiveCell.Address
            End If
        ElseIf TypeName(Application.ActiveSheet) = "Chart" Then
            sDefault = " first select a Worksheet"
        Else
            sDefault = " Select Cell(s) or type address"
        End If
    End If
    Set rInput = Nothing
    For nAttempt = 1 To 3
        vReturn = False
        vReturn = Application.InputBox(sPrompt, sTitle, sDefault, x, y, Type:=0)
        If False = vReturn Or Len(vReturn) = 0 Then
            Exit For
        Else
            sAddr = vReturn
```

```
                If Left$(sAddr, 1) = "=" Then sAddr = Mid$(sAddr, 2, 256)
                If Left$(sAddr, 1) = Chr(34) Then sAddr = Mid$(sAddr, 2, 255)
                If Right$(sAddr, 1) = Chr(34) Then sAddr = Left$(sAddr, Len(sAddr) - 1)
                Set rInput = Application.Range(sAddr)
                If rInput Is Nothing Then
                    sAddr = Application.ConvertFormula(sAddr, xlR1C1, xlA1)
                    Set rInput = Application.Range(sAddr)
                    bGotRng = Not rInput Is Nothing
                Else
                    bGotRng = True
                End If
            End If
            If bGotRng Then
                If bActivate Then
                    On Error GoTo ErrH
                    bEvents = Application.EnableEvents
                    Application.EnableEvents = False
                    If Not Application.ActiveWorkbook Is rInput.Parent.Parent Then
                        rInput.Parent.Parent.Activate
                    End If
                    If Not Application.ActiveSheet Is rInput.Parent Then
                        rInput.Parent.Activate
                    End If
                    rInput.Select
                End If
                Exit For
            ElseIf nAttempt < 3 Then
                If MsgBox("Invalid reference, do you want to try again ?", _
                        vbOKCancel, sTitle) <> vbOK Then
                    Exit For
                End If
            End If
        Next
    cleanup:
        On Error Resume Next
        If bEvents Then
            Application.EnableEvents = True
        End If
        GetInputRange = bGotRng
        Exit Function
    ErrH:
        Set rInput = Nothing
        bGotRng = False
        Resume cleanup
    End Function
    Sub ArrayToRange1d(arr As Variant, Horizontal As Boolean, Optional rng As Range)                    ...
        If ArrayDimensions(arr) <> 1 Then Exit Sub
        If rng Is Nothing Then
            If GetInputRange(rng, "select range", "select range") = False Then Exit Sub
        End If
        Dim dif As Long, difC As Long
        dif = IIf(LBound(arr) = 0, 1, 0)
        If Horizontal Then
            rng.RESIZE(, UBound(arr) + dif) = arr
        Else
            rng.RESIZE(UBound(arr) + dif) = WorksheetFunction.Transpose(arr)
        End If
        rng.TextToColumns rng, , , , , , True
    End Sub
    Function WorksheetExists(SheetName As String, TargetWorkbook As Workbook) As Boolean                ...
        Dim TargetWorksheet As Worksheet
        On Error Resume Next
        Set TargetWorksheet = TargetWorkbook.SHEETS(SheetName)
        On Error GoTo 0
        WorksheetExists = Not TargetWorksheet Is Nothing
    End Function
    Function LargestLength(Optional myObj) As Long                                                      ...
        LargestLength = 0
        Dim Element     As Variant
        If IsMissing(myObj) Then
            If TypeName(Selection) = "Range" Then
                Set myObj = Selection
            Else
                Exit Function
            End If
        End If
        Select Case TypeName(myObj)
            Case Is = "String"
                LargestLength = Len(myObj)
            Case "Collection"
                For Each Element In myObj
                    If Len(Element) > LargestLength Then LargestLength = Len(Element)
                Next Element
            Case "Variant", "Variant()", "String()"
                For Element = LBound(myObj) To UBound(myObj)
                    If Len(myObj(Element)) > LargestLength Then LargestLength = Len(myObj(Element))
                Next
            Case Else
        End Select
    End Function
    Public Function Combine2Array(ByVal arr1 As Variant, ByVal arr2 As Variant) As Variant                    ...
        Dim LowRowArr1  As Long
        Dim HighRowArr1 As Long
        Dim LowColumnArr1 As Long
        Dim HighColumnArr1 As Long
        Dim NumOfRowsArr1 As Long
        Dim NumOfColumnsArr1 As Long
        Dim LowRowArr2  As Long
        Dim HighRowArr2 As Long
        Dim LowColumnArr2 As Long
        Dim HighColumnArr2 As Long
        Dim NumOfRowsArr2 As Long
        Dim NumOfColumnsArr2 As Long
        Dim Output      As Variant
        Dim OutputRow   As Long
        Dim OutputColumn As Long
        Dim RowIdx      As Long
        Dim ColIdx      As Long
        If (IsArray(arr1) = False) Or (IsArray(arr2) = False) Then
            Combine2Array = Null
            MsgBox "Both need to be array"
            Exit Function
        End If
        If (NumberOfArrayDimensions(arr1) <> 2) Or (NumberOfArrayDimensions(arr2) <> 2) Then
            Combine2Array = Null
            MsgBox "Both need to be 2D array"
            Exit Function
        End If
        LowRowArr1 = LBound(arr1, 1)
        HighRowArr1 = UBound(arr1, 1)
        LowColumnArr1 = LBound(arr1, 2)
        HighColumnArr1 = UBound(arr1, 2)
        NumOfRowsArr1 = HighRowArr1 - LowRowArr1 + 1
        NumOfColumnsArr1 = HighColumnArr1 - LowColumnArr1 + 1
        LowRowArr2 = LBound(arr2, 1)
        HighRowArr2 = UBound(arr2, 1)
        LowColumnArr2 = LBound(arr2, 2)
        HighColumnArr2 = UBound(arr2, 2)
        NumOfRowsArr2 = HighRowArr2 - LowRowArr2 + 1
        NumOfColumnsArr2 = HighColumnArr2 - LowColumnArr2 + 1
        If NumOfColumnsArr1 <> NumOfColumnsArr2 Then
            Combine2Array = Null
            MsgBox "Both array must have same number of column"
            Exit Function
        End If
        ReDim Output(0 To NumOfRowsArr1 + NumOfRowsArr2 - 1, 0 To NumOfColumnsArr1 - 1)
        For RowIdx = LowRowArr1 To HighRowArr1
            OutputColumn = 0
            For ColIdx = LowColumnArr1 To HighColumnArr1
                Output(OutputRow, OutputColumn) = arr1(RowIdx, ColIdx)
                OutputColumn = OutputColumn + 1
            Next ColIdx
            OutputRow = OutputRow + 1
        Next RowIdx
        For RowIdx = LowRowArr2 To HighRowArr2
            OutputColumn = 0
            For ColIdx = LowColumnArr2 To HighColumnArr2
                Output(OutputRow, OutputColumn) = arr2(RowIdx, ColIdx)
                OutputColumn = OutputColumn + 1
            Next ColIdx
            OutputRow = OutputRow + 1
        Next RowIdx
        Combine2Array = Output
    End Function
    Public Function NumberOfArrayDimensions(arr As Variant) As Byte                            ...
        Dim Ndx         As Byte
        Dim Res         As Long
        On Error Resume Next
        Do
            Ndx = Ndx + 1
            Res = UBound(arr, Ndx)
        Loop Until Err.Number <> 0
        NumberOfArrayDimensions = Ndx - 1
    End Function
    Sub TxtOverwrite(sFile As String, sText As String)                                         ...
        On Error GoTo ERR_HANDLER
        Dim FileNumber  As Integer
        FileNumber = FreeFile
        Open sFile For Output As #FileNumber
        Print #FileNumber, sText
        Close #FileNumber
    Exit_Err_Handler:
        Exit Sub
    ERR_HANDLER:
        MsgBox "The following error has occurred" & vbCrLf & vbCrLf & _
                "Error Number: " & Err.Number & vbCrLf & _
                "Error Source: TxtOverwrite" & vbCrLf & _
                "Error Description: " & Err.Description, vbCritical, "An Error has Occurred!"
        GoTo Exit_Err_Handler
    End Sub
    Sub FollowLink(FolderPath As String)                                                       ...
        If Right(FolderPath, 1) = "\" Then FolderPath = Left(FolderPath, Len(FolderPath) - 1)
        On Error Resume Next
        Dim oShell      As Object
        Dim Wnd         As Object
        Set oShell = CreateObject("Shell.Application")
        For Each Wnd In oShell.Windows
            If Wnd.Name = "File Explorer" Then
                If Wnd.document.Folder.Self.Path = FolderPath Then Exit Sub
            End If
        Next Wnd
        Application.ThisWorkbook.FollowHyperlink Address:=FolderPath, NewWindow:=True
    End Sub
    Function TxtRead(sPath As Variant) As String                                               ...
        Dim sTXT        As String
        If Dir(sPath) = "" Then
            Debug.Print "File was not found."
            Debug.Print sPath
            Exit Function
        End If
        Open sPath For Input As #1
        Do Until EOF(1)
            Line Input #1, sTXT
            TxtRead = TxtRead & sTXT & vbLf
        Loop
        Close
        If Len(TxtRead) = 0 Then
            TxtRead = ""
        Else
            TxtRead = Left(TxtRead, Len(TxtRead) - 1)
        End If
    End Function
    Public Function ArrayRemoveEmptyElements(varArray As Variant) As Variant                   ...
        Dim TempArray() As Variant
        Dim OldIndex    As Integer
        Dim NewIndex    As Integer
        ReDim TempArray(LBound(varArray) To UBound(varArray))
        For OldIndex = LBound(varArray) To UBound(varArray)
            If Not Trim(varArray(OldIndex) & " ") = "" Then
                TempArray(NewIndex) = varArray(OldIndex)
                NewIndex = NewIndex + 1
            End If
        Next OldIndex
        ReDim Preserve TempArray(LBound(varArray) To NewIndex - 1)
        ArrayRemoveEmptyElements = TempArray
        varArray = TempArray
    End Function
    Public Function FileExists(ByVal filename As String) As Boolean                    ...
        If InStr(1, filename, "\") = 0 Then Exit Function
        If Right(filename, 1) = "\" Then filename = Left(filename, Len(filename) - 1)
        FileExists = (Dir(filename, vbArchive + vbHidden + vbReadOnly + vbSystem) <> "")
    End Function
    Function ArrayTrim(ByVal arr As Variant)                                            ...
        Dim i           As Long
        For i = LBound(arr) To UBound(arr)
            arr(i) = Trim(arr(i))
        Next
        ArrayTrim = arr
    End Function
```

--- Printer ---

```
Option Explicit
Public Sub dp(var As Variant)
    Dim Element     As Variant
    Dim i           As Long
    Select Case TypeName(var)
        Case Is = "String", "Long", "Integer", "Double", "Boolean"
            Debug.Print var
        Case Is = "Variant()", "String()", "Long()", "Integer()"
            printArray var
        Case Is = "Collection"
            printCollection var
        Case Is = "Dictionary"
            printDictionary var
        Case Is = "Range"
            printRange var
        Case Is = "Date"
            Debug.Print var
        Case Is = "IXMLDOMElement"
            PrintXML var
        Case Else
    End Select
End Sub
Sub PrintXML(var)
    Debug.Print var.xml
End Sub
Public Sub printRange(var As Variant)
    If var.Areas.Count = 1 Then
        dp var.Value
    Else
        Dim out     As Variant
        Dim Temp    As Variant
        Dim i       As Long
        For i = 1 To var.Areas.Count
            Temp = var.Areas(i).Value
            If IsEmpty(out) Then
                out = Temp
            Else
                out = Combine2Array(out, Temp)
            End If
        Next
        dp out
    End If
End Sub
Private Sub printArray(var As Variant)
    Dim Element
    If ArrayDimensions(var) = 1 Then
        For Each Element In var
            dp Element
        Next
    ElseIf ArrayDimensions(var) > 1 Then
        DPH var
    End If
End Sub
Private Sub printCollection(var As Variant)
    Dim elem        As Variant
    For Each elem In var
        dp elem
    Next elem
End Sub
Private Sub printDictionary(var As Variant)
    Dim i As Long: Dim iCount As Long
    Dim arrKeys
    Dim sKey        As String
    Dim varItem
    Dim key         As Variant
    For Each key In var.Keys
        dp var(key)
    Next key
End Sub
Private Sub DPH(ByVal Hairetu, Optional HyoujiMaxNagasa%, Optional HairetuName$)
```

```
    Call DebugPrintHairetu(Hairetu, HyoujiMaxNagasa, HairetuName)
End Sub
Public Function ArrayDimensions(ByVal vArray As Variant) As Long                                                   ...
    Dim dimnum      As Long
    Dim ErrorCheck  As Long
    On Error GoTo FinalDimension
    For dimnum = 1 To 60000
        ErrorCheck = LBound(vArray, dimnum)
    Next
FinalDimension:
    ArrayDimensions = dimnum - 1
End Function
Private Sub DebugPrintHairetu(ByVal Hairetu, Optional HyoujiMaxNagasa%, Optional HairetuName$)                     ...
    Dim i&, j&, k&, M&, n&
    Dim TateMin&, TateMax&, YokoMin&, YokoMax&
    Dim WithTableHairetu
    Dim NagasaList, MaxNagasaList
    Dim NagasaOnajiList
    Dim OutputList
    Const SikiriMoji$ = "|"
    Dim Jigen2%
    On Error Resume Next
    Jigen2 = UBound(Hairetu, 2)
    On Error GoTo 0
    If Jigen2 = 0 Then
        Hairetu = Application.Transpose(Hairetu)
    End If
    TateMin = LBound(Hairetu, 1)
    TateMax = UBound(Hairetu, 1)
    YokoMin = LBound(Hairetu, 2)
    YokoMax = UBound(Hairetu, 2)
    ReDim WithTableHairetu(1 To TateMax - TateMin + 1 + 1, 1 To YokoMax - YokoMin + 1 + 1)
    For i = 1 To TateMax - TateMin + 1
        WithTableHairetu(i + 1, 1) = TateMin + i - 1
        For j = 1 To YokoMax - YokoMin + 1
            WithTableHairetu(1, j + 1) = YokoMin + j - 1
            WithTableHairetu(i + 1, j + 1) = Hairetu(i - 1 + TateMin, j - 1 + YokoMin)
        Next j
    Next i
    n = UBound(WithTableHairetu, 1)
    M = UBound(WithTableHairetu, 2)
    ReDim NagasaList(1 To n, 1 To M)
    ReDim MaxNagasaList(1 To M)
    Dim tmpStr$
    For j = 1 To M
        For i = 1 To n
            If j > 1 And HyoujiMaxNagasa <> 0 Then
                tmpStr = WithTableHairetu(i, j)
                WithTableHairetu(i, j) = ShortenToByteCharacters(tmpStr, HyoujiMaxNagasa)
            End If
            NagasaList(i, j) = LenB(StrConv(WithTableHairetu(i, j), vbFromUnicode))
            MaxNagasaList(j) = WorksheetFunction.Max(MaxNagasaList(j), NagasaList(i, j))
        Next i
    Next j
    ReDim NagasaOnajiList(1 To n, 1 To M)
    Dim TmpMaxNagasa&
    For j = 1 To M
        TmpMaxNagasa = MaxNagasaList(j)
        For i = 1 To n
            NagasaOnajiList(i, j) = WithTableHairetu(i, j) & WorksheetFunction.Rept(" ",  _
            TmpMaxNagasa - NagasaList(i, j))
        Next i
    Next j
    ReDim OutputList(1 To n)
    For i = 1 To n
        For j = 1 To M
            If j = 1 Then
                OutputList(i) = NagasaOnajiList(i, j)
            Else
                OutputList(i) = OutputList(i) & SikiriMoji & NagasaOnajiList(i, j)
            End If
        Next j
    Next i
    Debug.Print HairetuName
    For i = 1 To n
        Debug.Print OutputList(i)
    Next i
End Sub
Public Function ShortenToByteCharacters(Mojiretu$, ByteNum%)                                                    ...
    Dim OriginByte%
    Dim Output
    OriginByte = LenB(StrConv(Mojiretu, vbFromUnicode))
    If OriginByte <= ByteNum Then
        Output = Mojiretu
    Else
        Dim RuikeiByteList, BunkaiMojiretu
        RuikeiByteList = CalculateByteCharacters(Mojiretu)
        BunkaiMojiretu = TextDecomposition(Mojiretu)
        Dim AddMoji$
        AddMoji = "."
        Dim i&, n&
        n = Len(Mojiretu)
        For i = 1 To n
            If RuikeiByteList(i) < ByteNum Then
                Output = Output & BunkaiMojiretu(i)
            ElseIf RuikeiByteList(i) = ByteNum Then
                If LenB(StrConv(BunkaiMojiretu(i), vbFromUnicode)) = 1 Then
                    Output = Output & AddMoji
                Else
                    Output = Output & AddMoji & AddMoji
                End If
                Exit For
            ElseIf RuikeiByteList(i) > ByteNum Then
                Output = Output & AddMoji
                Exit For
            End If
        Next i
    End If
    ShortenToByteCharacters = Output
End Function
Private Function CalculateByteCharacters(Mojiretu$)                                                             ...
    Dim MojiKosu%
    MojiKosu = Len(Mojiretu)
    Dim Output
    ReDim Output(1 To MojiKosu)
    Dim i&
    Dim TmpMoji$
    For i = 1 To MojiKosu
        TmpMoji = Mid(Mojiretu, i, 1)
        If i = 1 Then
            Output(i) = LenB(StrConv(TmpMoji, vbFromUnicode))
        Else
            Output(i) = LenB(StrConv(TmpMoji, vbFromUnicode)) + Output(i - 1)
        End If
    Next i
    CalculateByteCharacters = Output
End Function
Private Function TextDecomposition(Mojiretu$)                                                                   ...
    Dim i&, n&
    Dim Output
    n = Len(Mojiretu)
    ReDim Output(1 To n)
    For i = 1 To n
        Output(i) = Mid(Mojiretu, i, 1)
    Next i
    TextDecomposition = Output
End Function
Function DpHeader( _                                                                                            ...
        str As Variant, _
        Optional lvl As Integer = 1, _
        Optional Character As String = "'", _
        Optional Top As Boolean, _
        Optional Bottom As Boolean) As String
    If lvl < 1 Then lvl = 1
```

```
        If Character = "" Then Character = "'"
        Dim indentation As Integer
        indentation = (lvl * 4) - 4 + 1
        Dim quote       As String: quote = "'"
        Dim S           As String
        Dim Element     As Variant
        If Top = True Then S = vbNewLine & quote & String(indentation + LargestLength(str), Character)  _
        & vbNewLine
        If TypeName(str) <> "String" Then
            For Each Element In str
                S = S & quote & Character & Space(indentation) & Element & vbNewLine
            Next
        Else
            S = S & quote & String(indentation, Character) & str
        End If
        If Bottom = True Then S = S & quote & String(indentation + LargestLength(str), Character)
        DpHeader = S
    End Function
```

--- A_TESTS ---

```
Option Explicit
Sub TestRegistryEditor()
    Dim r           As New RegistryEditor
    r.VBA_OpenSettings
    Stop
    r.VBA_SaveSetting "TestAppName", "TestSectionName", "TestKeyName", "TestValue"
    dp r.VBA_GetSetting("TestAppName", "TestSectionName", "TestKeyName")
    dp String(20, "~")
    Stop
    r.VBA_SaveSetting "TestAppName", "TestSectionName", "TestKeyName", "NewValue"
    dp r.VBA_GetSetting("TestAppName", "TestSectionName", "TestKeyName")
    dp String(20, "~")
    Stop
    r.VBA_DeleteSetting "TestAppName", "TestSectionName", "TestKeyName"
    dp r.VBA_GetSetting("TestAppName", "TestSectionName", "TestKeyName")
    dp String(20, "~")
    Stop
    Dim i           As Long
    For i = 1 To 5
        r.VBA_SaveSetting "TestAppName", "TestSectionName", "TestKeyName" & i, "TestValue" & i
    Next
    dp r.VBA_GetAllSettings("TestAppName", "TestSectionName")
    Stop
    For i = 1 To 5
        r.VBA_DeleteSetting "TestAppName", "TestSectionName", "TestKeyName" & i
    Next
End Sub
Sub TestINI()
    Dim FilePath    As String: FilePath = ThisWorkbook.Path & "\test.INI"
    IniWrite FilePath, "Settings1", "KeyName1", "Value1"
    IniWrite FilePath, "Settings1", "KeyName2", "2"
    IniWrite FilePath, "Settings1", "KeyName3", "3"
    Stop
    IniWrite FilePath, "Settings1", "KeyName1", "Updated Value"
    Stop
    Dim i           As Long
    For i = 1 To 5
        IniWrite FilePath, "Settings" & i, "KeyName" & i, i
    Next
    Stop
    dp String(5, "~") & " Printing sections of " & FilePath
    dp IniSections(FilePath)
    Stop
    dp String(5, "~") & " Printing keys of section Settings1"
    dp IniSectionKeys(FilePath, "Settings1")
    Stop
    dp String(5, "~") & " Printing all lines of section Settings1"
    dp IniReadSection(FilePath, "Settings1")
    Stop
    dp String(5, "~") & " Printing value of Section: Settings1, Keyname: Keyname1"
    dp IniReadKey(FilePath, "Settings1", "KeyName1")
End Sub
Sub TestSettingsTable()
    Dim t           As New aSettingsTable
    t.AddOrModify "TestApp", "TestSection", "TestKey", "TestValue"
    dp "Added value: " & t.Value("TestApp", "TestSection", "TestKey")
    dp String(20, "~")
    t.AddOrModify "TestApp", "TestSection", "TestKey", "NewValue"
    dp "Modified Value: " & t.Value("TestApp", "TestSection", "TestKey")
    dp String(20, "~")
    Dim i           As Long
    For i = 1 To 5
        t.AddOrModify "TestApp" & i, "TestSection" & i, "TestKey" & i, "TestValue" & i
    Next
    dp t.toINI("TestApp")
    dp String(20, "~")
    dp t.toTreeviewArray("TestApp")
    dp String(20, "~")
    dp t.toXML(aSettingsTable.Apps(1))
End Sub
```

```
Public Sub TestJson()                                                                                    ...
    Dim JsonText    As String
    JsonText = TxtRead(JSONTestFile)
    dp JsonToINI(JsonText)
    Debug.Print
    dp JsonToTable("test", JsonText)
    Debug.Print
    dp JsonToTreeviewArray("test", JsonText)
End Sub
```

--- JsonConverter ---

```
Option Explicit
#If Mac Then
#If VBA7 Then
Private Declare PtrSafe Function utc_popen Lib "/usr/lib/libc.dylib" Alias "popen" _
        (ByVal utc_Command As String, ByVal utc_Mode As String) As LongPtr
Private Declare PtrSafe Function utc_pclose Lib "/usr/lib/libc.dylib" Alias "pclose" _
        (ByVal utc_File As LongPtr) As LongPtr
Private Declare PtrSafe Function utc_fread Lib "/usr/lib/libc.dylib" Alias "fread" _
        (ByVal utc_Buffer As String, ByVal utc_Size As LongPtr, ByVal utc_Number As LongPtr, ByVal  _
        utc_File As LongPtr) As LongPtr
Private Declare PtrSafe Function utc_feof Lib "/usr/lib/libc.dylib" Alias "feof" _
        (ByVal utc_File As LongPtr) As LongPtr
#Else
Private Declare Function utc_popen Lib "libc.dylib" Alias "popen" _
        (ByVal utc_Command As String, ByVal utc_Mode As String) As Long
Private Declare Function utc_pclose Lib "libc.dylib" Alias "pclose" _
        (ByVal utc_File As Long) As Long
Private Declare Function utc_fread Lib "libc.dylib" Alias "fread" _
        (ByVal utc_Buffer As String, ByVal utc_Size As Long, ByVal utc_Number As Long, ByVal  _
        utc_File As Long) As Long
Private Declare Function utc_feof Lib "libc.dylib" Alias "feof" _
        (ByVal utc_File As Long) As Long
#End If
#ElseIf VBA7 Then
Private Declare PtrSafe Function utc_GetTimeZoneInformation Lib "kernel32" Alias  _
"GetTimeZoneInformation" _
        (utc_lpTimeZoneInformation As utc_TIME_ZONE_INFORMATION) As Long
Private Declare PtrSafe Function utc_SystemTimeToTzSpecificLocalTime Lib "kernel32" Alias  _
"SystemTimeToTzSpecificLocalTime" _
        (utc_lpTimeZoneInformation As utc_TIME_ZONE_INFORMATION, utc_lpUniversalTime As  _
        utc_SYSTEMTIME, utc_lpLocalTime As utc_SYSTEMTIME) As Long
Private Declare PtrSafe Function utc_TzSpecificLocalTimeToSystemTime Lib "kernel32" Alias  _
"TzSpecificLocalTimeToSystemTime" _
        (utc_lpTimeZoneInformation As utc_TIME_ZONE_INFORMATION, utc_lpLocalTime As utc_SYSTEMTIME,  _
        utc_lpUniversalTime As utc_SYSTEMTIME) As Long
#Else
Private Declare Function utc_GetTimeZoneInformation Lib "kernel32" Alias "GetTimeZoneInformation" _
        (utc_lpTimeZoneInformation As utc_TIME_ZONE_INFORMATION) As Long
Private Declare Function utc_SystemTimeToTzSpecificLocalTime Lib "kernel32" Alias  _
"SystemTimeToTzSpecificLocalTime" _
        (utc_lpTimeZoneInformation As utc_TIME_ZONE_INFORMATION, utc_lpUniversalTime As  _
        utc_SYSTEMTIME, utc_lpLocalTime As utc_SYSTEMTIME) As Long
Private Declare Function utc_TzSpecificLocalTimeToSystemTime Lib "kernel32" Alias  _
"TzSpecificLocalTimeToSystemTime" _
        (utc_lpTimeZoneInformation As utc_TIME_ZONE_INFORMATION, utc_lpLocalTime As utc_SYSTEMTIME,  _
        utc_lpUniversalTime As utc_SYSTEMTIME) As Long
#End If
#If Mac Then
#If VBA7 Then
Private Type utc_ShellResult
    utc_Output      As String
    utc_ExitCode    As LongPtr
End Type
#Else
Private Type utc_ShellResult
    utc_Output      As String
    utc_ExitCode    As Long
End Type
#End If
#Else
Private Type utc_SYSTEMTIME
    utc_wYear       As Integer
    utc_wMonth      As Integer
    utc_wDayOfWeek  As Integer
    utc_wDay        As Integer
    utc_wHour       As Integer
    utc_wMinute     As Integer
    utc_wSecond     As Integer
    utc_wMilliseconds As Integer
End Type
```

```
Private Type utc_TIME_ZONE_INFORMATION
    utc_Bias        As Long
    utc_StandardName(0 To 31) As Integer
    utc_StandardDate As utc_SYSTEMTIME
    utc_StandardBias As Long
    utc_DaylightName(0 To 31) As Integer
    utc_DaylightDate As utc_SYSTEMTIME
    utc_DaylightBias As Long
End Type
#End If
Private Type json_Options
    UseDoubleForLargeNumbers As Boolean
    AllowUnquotedKeys As Boolean
    EscapeSolidus   As Boolean
End Type
Public JsonOptions  As json_Options
Public Function ParseJson(ByVal JsonString As String) As Object                                    ...
    Dim json_Index  As Long
    json_Index = 1
    JsonString = VBA.Replace(VBA.Replace(VBA.Replace(JsonString, VBA.vbCr, ""), VBA.vbLf, ""), VBA. _
    vbTab, "")
    json_SkipSpaces JsonString, json_Index
    Select Case VBA.Mid$(JsonString, json_Index, 1)
        Case "{"
            Set ParseJson = json_ParseObject(JsonString, json_Index)
        Case "["
            Set ParseJson = json_ParseArray(JsonString, json_Index)
        Case Else
            Err.Raise 10001, "JSONConverter", json_ParseErrorMessage(JsonString, json_Index, "
    End Select
End Function
Public Function ConvertToJson(ByVal JsonValue As Variant, Optional ByVal Whitespace As Variant,  _      ...
Optional ByVal json_CurrentIndentation As Long = 0) As String
    Dim json_Buffer As String
    Dim json_BufferPosition As Long
    Dim json_BufferLength As Long
    Dim json_Index  As Long
    Dim json_LBound As Long
    Dim json_UBound As Long
    Dim json_IsFirstItem As Boolean
    Dim json_Index2D As Long
    Dim json_LBound2D As Long
    Dim json_UBound2D As Long
    Dim json_IsFirstItem2D As Boolean
    Dim json_Key    As Variant
    Dim json_Value  As Variant
    Dim json_DateStr As String
    Dim json_Converted As String
    Dim json_SkipItem As Boolean
    Dim json_PrettyPrint As Boolean
    Dim json_Indentation As String
    Dim json_InnerIndentation As String
    json_LBound = -1
    json_UBound = -1
    json_IsFirstItem = True
    json_LBound2D = -1
    json_UBound2D = -1
    json_IsFirstItem2D = True
    json_PrettyPrint = Not IsMissing(Whitespace)
    Select Case VBA.VarType(JsonValue)
        Case VBA.vbNull
            ConvertToJson = "null"
        Case VBA.vbDate
            json_DateStr = ConvertToIso(VBA.CDate(JsonValue))
            ConvertToJson = """" & json_DateStr & """"
        Case VBA.vbString
            If Not JsonOptions.UseDoubleForLargeNumbers And json_StringIsLargeNumber(JsonValue) Then
                ConvertToJson = JsonValue
            Else
                ConvertToJson = """" & json_Encode(JsonValue) & """"
            End If
        Case VBA.vbBoolean
```

```
            If JsonValue Then
                ConvertToJson = "true"
            Else
                ConvertToJson = "false"
            End If
        Case VBA.vbArray To VBA.vbArray + VBA.vbByte
            If json_PrettyPrint Then
                If VBA.VarType(Whitespace) = VBA.vbString Then
                    json_Indentation = VBA.String$(json_CurrentIndentation + 1, Whitespace)
                    json_InnerIndentation = VBA.String$(json_CurrentIndentation + 2, Whitespace)
                Else
                    json_Indentation = VBA.Space$((json_CurrentIndentation + 1) * Whitespace)
                    json_InnerIndentation = VBA.Space$((json_CurrentIndentation + 2) * Whitespace)
                End If
            End If
            json_BufferAppend json_Buffer, "[", json_BufferPosition, json_BufferLength
            On Error Resume Next
            json_LBound = LBound(JsonValue, 1)
            json_UBound = UBound(JsonValue, 1)
            json_LBound2D = LBound(JsonValue, 2)
            json_UBound2D = UBound(JsonValue, 2)
            If json_LBound >= 0 And json_UBound >= 0 Then
                For json_Index = json_LBound To json_UBound
                    If json_IsFirstItem Then
                        json_IsFirstItem = False
                    Else
                        json_BufferAppend json_Buffer, ",", json_BufferPosition, json_BufferLength
                    End If
                    If json_LBound2D >= 0 And json_UBound2D >= 0 Then
                        If json_PrettyPrint Then
                            json_BufferAppend json_Buffer, vbNewLine, json_BufferPosition,  _
                            json_BufferLength
                        End If
                        json_BufferAppend json_Buffer, json_Indentation & "[", json_BufferPosition,  _
                        json_BufferLength
                        For json_Index2D = json_LBound2D To json_UBound2D
                            If json_IsFirstItem2D Then
                                json_IsFirstItem2D = False
                            Else
                                json_BufferAppend json_Buffer, ",", json_BufferPosition,  _
                                json_BufferLength
                            End If
                            json_Converted = ConvertToJson(JsonValue(json_Index, json_Index2D),  _
                            Whitespace, json_CurrentIndentation + 2)
                            If json_Converted = "" Then
                                If json_IsUndefined(JsonValue(json_Index, json_Index2D)) Then
                                    json_Converted = "null"
                                End If
                            End If
                            If json_PrettyPrint Then
                                json_Converted = vbNewLine & json_InnerIndentation & json_Converted
                            End If
                            json_BufferAppend json_Buffer, json_Converted, json_BufferPosition,  _
                            json_BufferLength
                        Next json_Index2D
                        If json_PrettyPrint Then
                            json_BufferAppend json_Buffer, vbNewLine, json_BufferPosition,  _
                            json_BufferLength
                        End If
                        json_BufferAppend json_Buffer, json_Indentation & "]", json_BufferPosition,  _
                        json_BufferLength
                        json_IsFirstItem2D = True
                    Else
                        json_Converted = ConvertToJson(JsonValue(json_Index), Whitespace,  _
                        json_CurrentIndentation + 1)
                        If json_Converted = "" Then
                            If json_IsUndefined(JsonValue(json_Index)) Then
                                json_Converted = "null"
                            End If
                        End If
                        If json_PrettyPrint Then
                            json_Converted = vbNewLine & json_Indentation & json_Converted
                        End If
                        json_BufferAppend json_Buffer, json_Converted, json_BufferPosition,  _
                        json_BufferLength
                    End If
                Next json_Index
            End If
            On Error GoTo 0
            If json_PrettyPrint Then
                json_BufferAppend json_Buffer, vbNewLine, json_BufferPosition, json_BufferLength
                If VBA.VarType(Whitespace) = VBA.vbString Then
                    json_Indentation = VBA.String$(json_CurrentIndentation, Whitespace)
                Else
                    json_Indentation = VBA.Space$(json_CurrentIndentation * Whitespace)
                End If
            End If
            json_BufferAppend json_Buffer, json_Indentation & "]", json_BufferPosition,  _
            json_BufferLength
            ConvertToJson = json_BufferToString(json_Buffer, json_BufferPosition)
        Case VBA.vbObject
            If json_PrettyPrint Then
                If VBA.VarType(Whitespace) = VBA.vbString Then
                    json_Indentation = VBA.String$(json_CurrentIndentation + 1, Whitespace)
                Else
                    json_Indentation = VBA.Space$((json_CurrentIndentation + 1) * Whitespace)
                End If
            End If
            If VBA.TypeName(JsonValue) = "Dictionary" Then
                json_BufferAppend json_Buffer, "{", json_BufferPosition, json_BufferLength
                For Each json_Key In JsonValue.Keys
                    json_Converted = ConvertToJson(JsonValue(json_Key), Whitespace,  _
                    json_CurrentIndentation + 1)
                    If json_Converted = "" Then
                        json_SkipItem = json_IsUndefined(JsonValue(json_Key))
                    Else
                        json_SkipItem = False
                    End If
                    If Not json_SkipItem Then
                        If json_IsFirstItem Then
                            json_IsFirstItem = False
                        Else
                            json_BufferAppend json_Buffer, ",", json_BufferPosition,  _
                            json_BufferLength
                        End If
                        If json_PrettyPrint Then
                            json_Converted = vbNewLine & json_Indentation & """" & json_Key & """:  _
                            " & json_Converted
                        Else
                            json_Converted = """" & json_Key & """:" & json_Converted
                        End If
                        json_BufferAppend json_Buffer, json_Converted, json_BufferPosition,  _
                        json_BufferLength
                    End If
                Next json_Key
                If json_PrettyPrint Then
                    json_BufferAppend json_Buffer, vbNewLine, json_BufferPosition, json_BufferLength
                    If VBA.VarType(Whitespace) = VBA.vbString Then
                        json_Indentation = VBA.String$(json_CurrentIndentation, Whitespace)
                    Else
                        json_Indentation = VBA.Space$(json_CurrentIndentation * Whitespace)
                    End If
                End If
                json_BufferAppend json_Buffer, json_Indentation & "}", json_BufferPosition,  _
                json_BufferLength
            ElseIf VBA.TypeName(JsonValue) = "Collection" Then
                json_BufferAppend json_Buffer, "[", json_BufferPosition, json_BufferLength
                For Each json_Value In JsonValue
                    If json_IsFirstItem Then
                        json_IsFirstItem = False
                    Else
                        json_BufferAppend json_Buffer, ",", json_BufferPosition, json_BufferLength
                    End If
                    json_Converted = ConvertToJson(json_Value, Whitespace, json_CurrentIndentation  _
                    + 1)
                    If json_Converted = "" Then
                        If json_IsUndefined(json_Value) Then
                            json_Converted = "null"
                        End If
                    End If
                    If json_PrettyPrint Then
                        json_Converted = vbNewLine & json_Indentation & json_Converted
                    End If
                    json_BufferAppend json_Buffer, json_Converted, json_BufferPosition,  _
                    json_BufferLength
                Next json_Value
                If json_PrettyPrint Then
                    json_BufferAppend json_Buffer, vbNewLine, json_BufferPosition, json_BufferLength
                    If VBA.VarType(Whitespace) = VBA.vbString Then
                        json_Indentation = VBA.String$(json_CurrentIndentation, Whitespace)
                    Else
                        json_Indentation = VBA.Space$(json_CurrentIndentation * Whitespace)
                    End If
                End If
                json_BufferAppend json_Buffer, json_Indentation & "]", json_BufferPosition,  _
                json_BufferLength
            End If
            ConvertToJson = json_BufferToString(json_Buffer, json_BufferPosition)
        Case VBA.vbInteger, VBA.vbLong, VBA.vbSingle, VBA.vbDouble, VBA.vbCurrency, VBA.vbDecimal
            ConvertToJson = VBA.Replace(JsonValue, ",", ".")
        Case Else
            On Error Resume Next
            ConvertToJson = JsonValue
            On Error GoTo 0
    End Select
End Function
Private Function json_ParseObject(json_String As String, ByRef json_Index As Long) As dictionary          ...
    Dim json_Key     As String
    Dim json_NextChar As String
    Set json_ParseObject = New dictionary
    json_SkipSpaces json_String, json_Index
    If VBA.Mid$(json_String, json_Index, 1) <> "{" Then
        Err.Raise 10001, "JSONConverter", json_ParseErrorMessage(json_String, json_Index, "Expe
    Else
        json_Index = json_Index + 1
        Do
            json_SkipSpaces json_String, json_Index
            If VBA.Mid$(json_String, json_Index, 1) = "}" Then
                json_Index = json_Index + 1
                Exit Function
            ElseIf VBA.Mid$(json_String, json_Index, 1) = "," Then
                json_Index = json_Index + 1
                json_SkipSpaces json_String, json_Index
            End If
            json_Key = json_ParseKey(json_String, json_Index)
            json_NextChar = json_Peek(json_String, json_Index)
            If json_NextChar = "[" Or json_NextChar = "{" Then
                Set json_ParseObject.item(json_Key) = json_ParseValue(json_String, json_Index)
            Else
                json_ParseObject.item(json_Key) = json_ParseValue(json_String, json_Index)
            End If
        Loop
    End If
End Function
Private Function json_ParseArray(json_String As String, ByRef json_Index As Long) As Collection          ...
    Set json_ParseArray = New Collection
    json_SkipSpaces json_String, json_Index
    If VBA.Mid$(json_String, json_Index, 1) <> "[" Then
        Err.Raise 10001, "JSONConverter", json_ParseErrorMessage(json_String, json_Index, "Expe
    Else
        json_Index = json_Index + 1
        Do
            json_SkipSpaces json_String, json_Index
            If VBA.Mid$(json_String, json_Index, 1) = "]" Then
                json_Index = json_Index + 1
                Exit Function
            ElseIf VBA.Mid$(json_String, json_Index, 1) = "," Then
                json_Index = json_Index + 1
                json_SkipSpaces json_String, json_Index
            End If
            json_ParseArray.Add json_ParseValue(json_String, json_Index)
        Loop
    End If
End Function
Private Function json_ParseValue(json_String As String, ByRef json_Index As Long) As Variant                ...
    json_SkipSpaces json_String, json_Index
    Select Case VBA.Mid$(json_String, json_Index, 1)
        Case "{"
            Set json_ParseValue = json_ParseObject(json_String, json_Index)
        Case "["
            Set json_ParseValue = json_ParseArray(json_String, json_Index)
        Case """", "'"
            json_ParseValue = json_ParseString(json_String, json_Index)
        Case Else
            If VBA.Mid$(json_String, json_Index, 4) = "true" Then
                json_ParseValue = True
                json_Index = json_Index + 4
            ElseIf VBA.Mid$(json_String, json_Index, 5) = "false" Then
                json_ParseValue = False
                json_Index = json_Index + 5
            ElseIf VBA.Mid$(json_String, json_Index, 4) = "null" Then
                json_ParseValue = Null
                json_Index = json_Index + 4
            ElseIf VBA.InStr("+-0123456789", VBA.Mid$(json_String, json_Index, 1)) Then
                json_ParseValue = json_ParseNumber(json_String, json_Index)
            Else
                Err.Raise 10001, "JSONConverter", json_ParseErrorMessage(json_String, json_Index, "E
            End If
    End Select
End Function
Private Function json_ParseString(json_String As String, ByRef json_Index As Long) As String                ...
    Dim json_Quote  As String
    Dim json_Char   As String
    Dim json_Code   As String
    Dim json_Buffer As String
    Dim json_BufferPosition As Long
    Dim json_BufferLength As Long
    json_SkipSpaces json_String, json_Index
    json_Quote = VBA.Mid$(json_String, json_Index, 1)
    json_Index = json_Index + 1
    Do While json_Index > 0 And json_Index <= Len(json_String)
        json_Char = VBA.Mid$(json_String, json_Index, 1)
        Select Case json_Char
            Case "\"
                json_Index = json_Index + 1
                json_Char = VBA.Mid$(json_String, json_Index, 1)
                Select Case json_Char
                    Case """", "\", "/", "'"
                        json_BufferAppend json_Buffer, json_Char, json_BufferPosition,  _
                        json_BufferLength
                        json_Index = json_Index + 1
                    Case "b"
                        json_BufferAppend json_Buffer, vbBack, json_BufferPosition,  _
                        json_BufferLength
                        json_Index = json_Index + 1
                    Case "f"
                        json_BufferAppend json_Buffer, vbFormFeed, json_BufferPosition,  _
                        json_BufferLength
                        json_Index = json_Index + 1
                    Case "n"
                        json_BufferAppend json_Buffer, vbCrLf, json_BufferPosition,  _
                        json_BufferLength
                        json_Index = json_Index + 1
                    Case "r"
                        json_BufferAppend json_Buffer, vbCr, json_BufferPosition, json_BufferLength
                        json_Index = json_Index + 1
                    Case "t"
                        json_BufferAppend json_Buffer, vbTab, json_BufferPosition, json_BufferLength
                        json_Index = json_Index + 1
                    Case "u"
                        json_Index = json_Index + 1
                        json_Code = VBA.Mid$(json_String, json_Index, 4)
                        json_BufferAppend json_Buffer, VBA.ChrW(VBA.val("&h" + json_Code)),  _
                        json_BufferPosition, json_BufferLength
                        json_Index = json_Index + 4
                End Select
            Case json_Quote
                json_ParseString = json_BufferToString(json_Buffer, json_BufferPosition)
                json_Index = json_Index + 1
                Exit Function
            Case Else
                json_BufferAppend json_Buffer, json_Char, json_BufferPosition, json_BufferLength
                json_Index = json_Index + 1
        End Select
    Loop
End Function
Private Function json_ParseNumber(json_String As String, ByRef json_Index As Long) As Variant                    ...
    Dim json_Char   As String
    Dim json_Value  As String
    Dim json_IsLargeNumber As Boolean
    json_SkipSpaces json_String, json_Index
    Do While json_Index > 0 And json_Index <= Len(json_String)
        json_Char = VBA.Mid$(json_String, json_Index, 1)
        If VBA.InStr("+-0123456789.eE", json_Char) Then
            json_Value = json_Value & json_Char
            json_Index = json_Index + 1
        Else
            json_IsLargeNumber = IIf(InStr(json_Value, "."), Len(json_Value) >= 17, Len(json_Value)  _
            >= 16)
            If Not JsonOptions.UseDoubleForLargeNumbers And json_IsLargeNumber Then
                json_ParseNumber = json_Value
            Else
                json_ParseNumber = VBA.val(json_Value)
            End If
            Exit Function
        End If
    Loop
End Function
Private Function json_ParseKey(json_String As String, ByRef json_Index As Long) As String                      ...
    If VBA.Mid$(json_String, json_Index, 1) = """" Or VBA.Mid$(json_String, json_Index, 1) = "'"  _
    Then
        json_ParseKey = json_ParseString(json_String, json_Index)
    ElseIf JsonOptions.AllowUnquotedKeys Then
        Dim json_Char As String
        Do While json_Index > 0 And json_Index <= Len(json_String)
            json_Char = VBA.Mid$(json_String, json_Index, 1)
            If (json_Char <> " ") And (json_Char <> ":") Then
                json_ParseKey = json_ParseKey & json_Char
                json_Index = json_Index + 1
            Else
                Exit Do
            End If
        Loop
    Else
        Err.Raise 10001, "JSONConverter", json_ParseErrorMessage(json_String, json_Index, "Expecting
    End If
    json_SkipSpaces json_String, json_Index
    If VBA.Mid$(json_String, json_Index, 1) <> ":" Then
        Err.Raise 10001, "JSONConverter", json_ParseErrorMessage(json_String, json_Index, "Expe
    Else
        json_Index = json_Index + 1
    End If
End Function
Private Function json_IsUndefined(ByVal json_Value As Variant) As Boolean                                      ...
    Select Case VBA.VarType(json_Value)
        Case VBA.vbEmpty
            json_IsUndefined = True
        Case VBA.vbObject
            Select Case VBA.TypeName(json_Value)
                Case "Empty", "Nothing"
                json_IsUndefined = True
            End Select
    End Select
End Function
Private Function json_Encode(ByVal json_Text As Variant) As String                                    ...
    Dim json_Index  As Long
    Dim json_Char   As String
    Dim json_AscCode As Long
    Dim json_Buffer As String
    Dim json_BufferPosition As Long
    Dim json_BufferLength As Long
    For json_Index = 1 To VBA.Len(json_Text)
        json_Char = VBA.Mid$(json_Text, json_Index, 1)
        json_AscCode = VBA.AscW(json_Char)
        If json_AscCode < 0 Then
            json_AscCode = json_AscCode + 65536
        End If
        Select Case json_AscCode
            Case 34
                json_Char = "\"""
            Case 92
                json_Char = "\\"
            Case 47
                If JsonOptions.EscapeSolidus Then
                    json_Char = "\/"
                End If
            Case 8
                json_Char = "\b"
            Case 12
                json_Char = "\f"
            Case 10
                json_Char = "\n"
            Case 13
                json_Char = "\r"
            Case 9
                json_Char = "\t"
            Case 0 To 31, 127 To 65535
                json_Char = "\u" & VBA.Right$("0000" & VBA.Hex$(json_AscCode), 4)
        End Select
        json_BufferAppend json_Buffer, json_Char, json_BufferPosition, json_BufferLength
    Next json_Index
    json_Encode = json_BufferToString(json_Buffer, json_BufferPosition)
End Function
Private Function json_Peek(json_String As String, ByVal json_Index As Long, Optional  _            ...
json_NumberOfCharacters As Long = 1) As String
    json_SkipSpaces json_String, json_Index
    json_Peek = VBA.Mid$(json_String, json_Index, json_NumberOfCharacters)
End Function
Private Sub json_SkipSpaces(json_String As String, ByRef json_Index As Long)                          ...
    Do While json_Index > 0 And json_Index <= VBA.Len(json_String) And VBA.Mid$(json_String,  _
    json_Index, 1) = " "
        json_Index = json_Index + 1
    Loop
End Sub
Private Function json_StringIsLargeNumber(json_String As Variant) As Boolean                          ...
    Dim json_Length As Long
    Dim json_CharIndex As Long
    json_Length = VBA.Len(json_String)
    If json_Length >= 16 And json_Length <= 100 Then
        Dim json_CharCode As String
        json_StringIsLargeNumber = True
        For json_CharIndex = 1 To json_Length
            json_CharCode = VBA.Asc(VBA.Mid$(json_String, json_CharIndex, 1))
            Select Case json_CharCode
                Case 46, 48 To 57, 69, 101
                Case Else
                    json_StringIsLargeNumber = False
                    Exit Function
            End Select
        Next json_CharIndex
    End If
End Function
```

```
Private Function json_ParseErrorMessage(json_String As String, ByRef json_Index As Long,  _        ...
ErrorMessage As String)
    Dim json_StartIndex As Long
    Dim json_StopIndex As Long
    json_StartIndex = json_Index - 10
    json_StopIndex = json_Index + 10
    If json_StartIndex <= 0 Then
        json_StartIndex = 1
    End If
    If json_StopIndex > VBA.Len(json_String) Then
        json_StopIndex = VBA.Len(json_String)
    End If
    json_ParseErrorMessage = "Error parsing JSON:" & VBA.vbNewLine & _
            VBA.Mid$(json_String, json_StartIndex, json_StopIndex - json_StartIndex + 1) & VBA. _
            vbNewLine & _
            VBA.Space$(json_Index - json_StartIndex) & "^" & VBA.vbNewLine & _
            ErrorMessage
End Function
Private Sub json_BufferAppend(ByRef json_Buffer As String, _                                       ...
        ByRef json_Append As Variant, _
        ByRef json_BufferPosition As Long, _
        ByRef json_BufferLength As Long)
    Dim json_AppendLength As Long
    Dim json_LengthPlusPosition As Long
    json_AppendLength = VBA.Len(json_Append)
    json_LengthPlusPosition = json_AppendLength + json_BufferPosition
    If json_LengthPlusPosition > json_BufferLength Then
        Dim json_AddedLength As Long
        json_AddedLength = IIf(json_AppendLength > json_BufferLength, json_AppendLength,  _
        json_BufferLength)
        json_Buffer = json_Buffer & VBA.Space$(json_AddedLength)
        json_BufferLength = json_BufferLength + json_AddedLength
    End If
    Mid$(json_Buffer, json_BufferPosition + 1, json_AppendLength) = CStr(json_Append)
    json_BufferPosition = json_BufferPosition + json_AppendLength
End Sub
Private Function json_BufferToString(ByRef json_Buffer As String, ByVal json_BufferPosition As Long) _ ...
As String
    If json_BufferPosition > 0 Then
        json_BufferToString = VBA.Left$(json_Buffer, json_BufferPosition)
    End If
End Function
Public Function ParseUtc(utc_UtcDate As Date) As Date                                                 ...
    On Error GoTo utc_ErrorHandling
#If Mac Then
    ParseUtc = utc_ConvertDate(utc_UtcDate)
#Else
    Dim utc_TimeZoneInfo As utc_TIME_ZONE_INFORMATION
    Dim utc_LocalDate As utc_SYSTEMTIME
    utc_GetTimeZoneInformation utc_TimeZoneInfo
    utc_SystemTimeToTzSpecificLocalTime utc_TimeZoneInfo, utc_DateToSystemTime(utc_UtcDate),  _
    utc_LocalDate
    ParseUtc = utc_SystemTimeToDate(utc_LocalDate)
#End If
    Exit Function
utc_ErrorHandling:
    Err.Raise 10011, "UtcConverter.ParseUtc", "UTC parsing error: " & Err.Number & " - " & Err. _
    Description
End Function
Public Function ConvertToUtc(utc_LocalDate As Date) As Date                                           ...
    On Error GoTo utc_ErrorHandling
#If Mac Then
    ConvertToUtc = utc_ConvertDate(utc_LocalDate, utc_ConvertToUtc:=True)
#Else
    Dim utc_TimeZoneInfo As utc_TIME_ZONE_INFORMATION
    Dim utc_UtcDate As utc_SYSTEMTIME
    utc_GetTimeZoneInformation utc_TimeZoneInfo
    utc_TzSpecificLocalTimeToSystemTime utc_TimeZoneInfo, utc_DateToSystemTime(utc_LocalDate),  _
    utc_UtcDate
    ConvertToUtc = utc_SystemTimeToDate(utc_UtcDate)
#End If
    Exit Function
```

```
utc_ErrorHandling:
    Err.Raise 10012, "UtcConverter.ConvertToUtc", "UTC conversion error: " & Err.Number & " - " &  _
    Err.Description
End Function
Public Function ParseIso(utc_IsoString As String) As Date                                          ...
    On Error GoTo utc_ErrorHandling
    Dim utc_Parts() As String
    Dim utc_DateParts() As String
    Dim utc_TimeParts() As String
    Dim utc_OffsetIndex As Long
    Dim utc_HasOffset As Boolean
    Dim utc_NegativeOffset As Boolean
    Dim utc_OffsetParts() As String
    Dim utc_Offset  As Date
    utc_Parts = VBA.Split(utc_IsoString, "T")
    utc_DateParts = VBA.Split(utc_Parts(0), "-")
    ParseIso = VBA.DateSerial(VBA.CInt(utc_DateParts(0)), VBA.CInt(utc_DateParts(1)), VBA.CInt( _
    utc_DateParts(2)))
    If UBound(utc_Parts) > 0 Then
        If VBA.InStr(utc_Parts(1), "Z") Then
            utc_TimeParts = VBA.Split(VBA.Replace(utc_Parts(1), "Z", ""), ":")
        Else
            utc_OffsetIndex = VBA.InStr(1, utc_Parts(1), "+")
            If utc_OffsetIndex = 0 Then
                utc_NegativeOffset = True
                utc_OffsetIndex = VBA.InStr(1, utc_Parts(1), "-")
            End If
            If utc_OffsetIndex > 0 Then
                utc_HasOffset = True
                utc_TimeParts = VBA.Split(VBA.Left$(utc_Parts(1), utc_OffsetIndex - 1), ":")
                utc_OffsetParts = VBA.Split(VBA.Right$(utc_Parts(1), Len(utc_Parts(1)) -  _
                utc_OffsetIndex), ":")
                Select Case UBound(utc_OffsetParts)
                    Case 0
                        utc_Offset = TimeSerial(VBA.CInt(utc_OffsetParts(0)), 0, 0)
                    Case 1
                        utc_Offset = TimeSerial(VBA.CInt(utc_OffsetParts(0)), VBA.CInt( _
                        utc_OffsetParts(1)), 0)
                    Case 2
                        utc_Offset = TimeSerial(VBA.CInt(utc_OffsetParts(0)), VBA.CInt( _
                        utc_OffsetParts(1)), Int(VBA.val(utc_OffsetParts(2))))
                End Select
                If utc_NegativeOffset Then: utc_Offset = -utc_Offset
            Else
                utc_TimeParts = VBA.Split(utc_Parts(1), ":")
            End If
        End If
        Select Case UBound(utc_TimeParts)
            Case 0
                ParseIso = ParseIso + VBA.TimeSerial(VBA.CInt(utc_TimeParts(0)), 0, 0)
            Case 1
                ParseIso = ParseIso + VBA.TimeSerial(VBA.CInt(utc_TimeParts(0)), VBA.CInt( _
                utc_TimeParts(1)), 0)
            Case 2
                ParseIso = ParseIso + VBA.TimeSerial(VBA.CInt(utc_TimeParts(0)), VBA.CInt( _
                utc_TimeParts(1)), Int(VBA.val(utc_TimeParts(2))))
        End Select
        ParseIso = ParseUtc(ParseIso)
        If utc_HasOffset Then
            ParseIso = ParseIso - utc_Offset
        End If
    End If
    Exit Function
utc_ErrorHandling:
    Err.Raise 10013, "UtcConverter.ParseIso", "ISO 8601 parsing error for " & utc_IsoString & ": "   _
    & Err.Number & " - " & Err.Description
End Function
Public Function ConvertToIso(utc_LocalDate As Date) As String                                      ...
    On Error GoTo utc_ErrorHandling
    ConvertToIso = VBA.Format$(ConvertToUtc(utc_LocalDate), "yyyy-mm-ddTHH:mm:ss.000Z")
    Exit Function
utc_ErrorHandling:
    Err.Raise 10014, "UtcConverter.ConvertToIso", "ISO 8601 conversion error: " & Err.Number & " -  _
    " & Err.Description
End Function
#If Mac Then
Private Function utc_ConvertDate(utc_Value As Date, Optional utc_ConvertToUtc As Boolean = False)  _                    ...
As Date
    Dim utc_ShellCommand As String
    Dim utc_Result  As utc_ShellResult
    Dim utc_Parts() As String
    Dim utc_DateParts() As String
    Dim utc_TimeParts() As String
    If utc_ConvertToUtc Then
        utc_ShellCommand = "date -ur `date -jf '%Y-%m-%d
                "'" & VBA.Format$(utc_Value, "yyyy-mm-dd HH:mm:ss") & "' " & _
    Else
        utc_ShellCommand = "date -jf '%Y-%m-%d %H:
                "'" & VBA.Format$(utc_Value, "yyyy-mm-dd HH:mm:ss") & " +0000' " & _
                "+'%
    End If
    utc_Result = utc_ExecuteInShell(utc_ShellCommand)
    If utc_Result.utc_Output = "" Then
        Err.Raise 10015, "UtcConverter.utc_ConvertDate"
    Else
        utc_Parts = Split(utc_Result.utc_Output, " ")
        utc_DateParts = Split(utc_Parts(0), "-")
        utc_TimeParts = Split(utc_Parts(1), ":")
        utc_ConvertDate = DateSerial(utc_DateParts(0), utc_DateParts(1), utc_DateParts(2)) + _
                TimeSerial(utc_TimeParts(0), utc_TimeParts(1), utc_TimeParts(2))
    End If
End Function
Private Function utc_ExecuteInShell(utc_ShellCommand As String) As utc_ShellResult                                       ...
#If VBA7 Then
    Dim utc_File    As LongPtr
    Dim utc_Read    As LongPtr
#Else
    Dim utc_File    As Long
    Dim utc_Read    As Long
#End If
    Dim utc_Chunk   As String
    On Error GoTo utc_ErrorHandling
    utc_File = utc_popen(utc_ShellCommand, "r")
    If utc_File = 0 Then: Exit Function
    Do While utc_feof(utc_File) = 0
        utc_Chunk = VBA.Space$(50)
        utc_Read = CLng(utc_fread(utc_Chunk, 1, Len(utc_Chunk) - 1, utc_File))
        If utc_Read > 0 Then
            utc_Chunk = VBA.Left$(utc_Chunk, CLng(utc_Read))
            utc_ExecuteInShell.utc_Output = utc_ExecuteInShell.utc_Output & utc_Chunk
        End If
    Loop
utc_ErrorHandling:
    utc_ExecuteInShell.utc_ExitCode = CLng(utc_pclose(utc_File))
End Function
#Else
Private Function utc_DateToSystemTime(utc_Value As Date) As utc_SYSTEMTIME                                                ...
    utc_DateToSystemTime.utc_wYear = VBA.Year(utc_Value)
    utc_DateToSystemTime.utc_wMonth = VBA.Month(utc_Value)
    utc_DateToSystemTime.utc_wDay = VBA.Day(utc_Value)
    utc_DateToSystemTime.utc_wHour = VBA.Hour(utc_Value)
    utc_DateToSystemTime.utc_wMinute = VBA.Minute(utc_Value)
    utc_DateToSystemTime.utc_wSecond = VBA.Second(utc_Value)
    utc_DateToSystemTime.utc_wMilliseconds = 0
End Function
Private Function utc_SystemTimeToDate(utc_Value As utc_SYSTEMTIME) As Date                                                ...
    utc_SystemTimeToDate = DateSerial(utc_Value.utc_wYear, utc_Value.utc_wMonth, utc_Value.utc_wDay) _
    + _
            TimeSerial(utc_Value.utc_wHour, utc_Value.utc_wMinute, utc_Value.utc_wSecond)
End Function
#End If
```

--- JSON ---

```
Public Function JSONTestFile()
    JSONTestFile = ThisWorkbook.Path & "\" & "test.json"
End Function
Public Function JsonToINI(JsonText As String) As String
    Dim JSON: Set JSON = JsonConverter.ParseJson(JsonTex
    Dim i           As Long
    Dim Section
    Dim KeyValue
    Dim out         As String
    For Each Section In JSON
        out = out & IIf(out <> "", vbNewLine, "") & "[" & Section & "]"
        For Each KeyValue In JSON(Sect
            out = out & vbNewLine & Space(4) & Join(KeyValue.Items, "=")
        Next
    Next
    JsonToINI = out
End Function
Public Function JsonToTable(App, JsonText As String) As Variant
    Dim JSON: Set JSON = JsonConverter.ParseJson(JsonText)
    Dim i           As Long
    Dim Section
    Dim KeyValue
    Dim counter
    Dim arr
    ReDim arr(1 To 1, 1 To 4)
    For Each Section In JSON
        If i > 0 Then ReDim Preserve arr(1 To UBound(arr, 1) + 1, 1 To 3)
        i = 0
        For Each KeyValue In JSON(Sect
            i = i + 1
            arr(UBound(arr, 1), 1) = App
            arr(UBound(arr, 1), 2) = Section
            arr(UBound(arr, 1), 3) = Split(Join(KeyValue.Items, "="), "=")(0)
            arr(UBound(arr, 1), 4) = Split(Join(KeyValue.Items, "="), "=")(1)
            arr = WorksheetFunction.Transpose(arr)
            If i < JSON(Section).Count Then
                ReDim Preserve arr(1 To 4, 1 To UBound(arr, 2) + 1)
            End If
            arr = WorksheetFunction.Transpose(arr)
        Next
    Next
    JsonToTable = arr
End Function
Public Function JsonToTreeviewArray(App, JsonText As String)
    Dim JSON: Set JSON = JsonConverter.ParseJson(JsonText)
    Dim Section
    Dim key
    Dim arr
    ReDim arr(1 To 1, 1 To 4)
    arr(1, 1) = App
    arr = WorksheetFunction.Transpose(arr)
    ReDim Preserve arr(1 To 4, 1 To UBound(arr, 2) + 1)
    arr = WorksheetFunction.Transpose(arr)
    Dim x, y
    For Each Section In JSON
        x = x + 1
        y = 0
        arr(UBound(arr, 1), 2) = Section
        arr = WorksheetFunction.Transpose(arr)
        ReDim Preserve arr(1 To 4, 1 To UBound(arr, 2) + 1)
        arr = WorksheetFunction.Transpose(arr)
        For Each key In JSON(Sect
            y = y + 1
            arr(UBound(arr, 1), 3) = Split(Join(key.Items, "="), "=")(0)
            arr = WorksheetFunction.Transpose(arr)
            ReDim Preserve arr(1 To 4, 1 To UBound(arr, 2) + 1)
            arr = WorksheetFunction.Transpose(arr)
            arr(UBound(arr, 1), 4) = Split(Join(key.Items, "="), "=")(1)
            If y < JSON(Section).Count Or x < JSON.Count Then
                arr = WorksheetFunction.Transpose(arr)
                ReDim Preserve arr(1 To 4, 1 To UBound(arr, 2) + 1)
                arr = WorksheetFunction.Transpose(arr)
            End If
        Next
    Next
    JsonToTreeviewArray = arr
End Function
```

--- aSettingsTable ---

```
Private Const SheetName = "Settings Table"
Private Const SettingsTableName = "tSettings"
Private SettingsTable As ListObject
Private SettingsSheet As Worksheet
Private Const colApp = 1
Private Const colSection = 2
Private Const colKey = 3
Private Const colValue = 4
Private Sub Class_Initialize()
    If Not WorksheetExists(SheetName, ThisWorkbook) Then CreateSheet
    Set SettingsSheet = ThisWorkbook.SHEETS(SheetName)
    Set SettingsTable = SettingsSheet.ListObjects(SettingsTableName)
End Sub
Private Sub CreateSheet()
    Application.ScreenUpdating = False
    Application.DisplayAlerts = False
    On Error Resume Next
    ThisWorkbook.SHEETS(SheetName).Delete
    On Error GoTo 0
    Dim ws          As Worksheet: Set ws = ThisWorkbook.SHEETS.Add(): ws.Name = SheetName
    Set SettingsSheet = ThisWorkbook.SHEETS(SheetName)
    Dim Headers     As Variant: Headers = Array("Application", "Section", "Key", "Value")
    ArrayToRange1d Headers, True, SettingsSheet.Range("A1")
    Set SettingsTable = SettingsSheet.ListObjects.Add(xlSrcRange, SettingsSheet.Range("A1"). _
    CurrentRegion, , xlYes)
    SettingsTable.Name = SettingsTableName
    Application.ScreenUpdating = True
    Application.DisplayAlerts = True
End Sub
Sub AddOrModify(App, Section, key, Value)
    Dim var: var = Array(App, Section, key, Value)
    With SettingsTable
        If Not IsArray(Filter(App, Section, key)) Then
            .ListRows.Add
            .ListRows(.ListRows.Count).Range.Value = var
        Else
            Dim i   As Long
            For i = 1 To .ListRows.Count
                If .ListRows(i).Range.Cells(colApp).Value = App _
                        And .ListRows(i).Range.Cells(colSection).Value = Section _
                        And .ListRows(i).Range.Cells(colKey).Value = key Then
                    .ListRows(i).Range.Cells(colValue).Value = Value
                End If
            Next
        End If
    End With
End Sub
Private Function Filter(App, Section, key)
    Dim var: var = Array(App, Section, key)
    Dim tVar
    tVar = ArrayFilter2d(SettingsTable.Range.Value, 1, App, True)
    tVar = ArrayFilter2d(tVar, 2, Section, False)
    tVar = ArrayFilter2d(tVar, 3, key, False)
    Filter = tVar
End Function
Public Function Apps() As Variant
    Dim arr
    arr = ArrayColumn(SettingsTable.DataBodyRange.Value, 1)
    Dim BA          As New BetterArray
    BA.Items = arr
    BA.Unique
    Apps = BA.Items
    Apps = WorksheetFunction.Transpose(Apps)
End Function
Public Function Sections(App)
    Dim arr: arr = ArrayFilter2d(SettingsTable.DataBodyRange.Value, colApp, App, False)
    arr = ArrayColumn(arr, colSection)
    Dim BA          As New BetterArray
    BA.Items = arr
    BA.Unique
    Sections = BA.Items
End Function
Public Function Keys(App, Section)                                                                   ...
    Dim arr
    arr = ArrayFilter2d(SettingsTable.DataBodyRange.Value, colApp, App, False)
    arr = ArrayFilter2d(arr, colSection, Section, False)
    Keys = ArrayColumn(arr, colKey)
End Function
Public Function Value(App, Section, key)                                                             ...
    Value = Filter(App, Section, key)(1, colValue)
End Function
Function toTreeviewArray(App)                                                                        ...
    Dim arr
    ReDim arr(1 To SettingsTable.ListColumns.Count, 1 To 1)
    Dim i           As Long
    Dim Section
    Dim key
    arr(1, UBound(arr, 2)) = App
    For Each Section In Sections(App)
        ReDim Preserve arr(1 To UBound(arr, 1), 1 To UBound(arr, 2) + 1)
        arr(2, UBound(arr, 2)) = Section
        For Each key In Keys(App, Section)
            ReDim Preserve arr(1 To UBound(arr, 1), 1 To UBound(arr, 2) + 1)
            arr(3, UBound(arr, 2)) = key
            ReDim Preserve arr(1 To UBound(arr, 1), 1 To UBound(arr, 2) + 1)
            arr(4, UBound(arr, 2)) = Value(App, Section, key)
        Next
    Next
    toTreeviewArray = WorksheetFunction.Transpose(arr)
End Function
Function toINI(App)                                                                                  ...
    Dim Section
    Dim key
    Dim out         As String
    For Each Section In Sections(App)
        out = out & IIf(out <> "", vbNewLine, "") & "[" & Section & "]"
        For Each key In Keys(App, Section)
            out = out & IIf(out <> "", vbNewLine, "") & Space(4) & key & "=" & Value(App, Section,  _
            key)
        Next
    Next
    toINI = out
End Function
Function toXML(App)                                                                                  ...
    Dim Section
    Dim key
    Dim var
    Dim out         As String
    Dim indentation As Long
    out = out & IIf(out <> "", vbNewLine, "") & "<" & App & ">"
    For Each Section In Sections(App)
        out = out & IIf(out <> "", vbNewLine, "") & Space(4) & "<" & Section & ">"
        var = Keys(App, Section)
        For Each key In var
            out = out & IIf(out <> "", vbNewLine, "") & Space(8) & "<" & key & ">" & Value(App,  _
            Section, key) & "</" & key & ">"
            If key = var(UBound(var)) Then
                out = out & IIf(out <> "", vbNewLine, "") & Space(4) & "</" & Section & ">"
            End If
        Next
    Next
    out = out & IIf(out <> "", vbNewLine, "") & "</" & App & ">"
    toXML = out
End Function
```

--- RegistryEditor ---

```
Option Explicit
Option Base 1
Option Compare Text
Private m_AppErr    As ApplicationError
Private Const C_NAME As String = "RegistryEditor.cls"
Private Type RegValue
    valueName       As String
    valueValue      As Variant
End Type
Const C_ERR_OFFSET = 0
Private Const C_ERR_NO_ERROR As Long = 0
Private Const C_ERR_INVALID_BASE_KEY As Long = C_ERR_OFFSET + vbObjectError + 1
Private Const C_ERR_INVALID_DATA_TYPE As Long = C_ERR_OFFSET + vbObjectError + 2
Private Const C_ERR_KEY_NOT_FOUND As Long = C_ERR_OFFSET + vbObjectError + 3
Private Const C_ERR_VALUE_NOT_FOUND As Long = C_ERR_OFFSET + vbObjectError + 4
Private Const C_ERR_DATA_TYPE_MISMATCH As Long = C_ERR_OFFSET + vbObjectError + 5
Private Const C_ERR_ENTRY_LOCKED As Long = C_ERR_OFFSET + vbObjectError + 6
Private Const C_ERR_INVALID_KEYNAME As Long = vbObjectError + C_ERR_OFFSET + 7
Private Const C_ERR_UNABLE_TO_OPEN_KEY As Long = vbObjectError + C_ERR_OFFSET + 8
Private Const C_ERR_UNABLE_TO_READ_KEY As Long = vbObjectError + C_ERR_OFFSET + 9
Private Const C_ERR_UNABLE_TO_CREATE_KEY As Long = vbObjectError + C_ERR_OFFSET + 10
Private Const C_ERR_UNABLE_TO_READ_VALUE As Long = vbObjectError + C_ERR_OFFSET + 11
Private Const C_ERR_UNABLE_TO_UPDATE_VALUE As Long = vbObjectError + C_ERR_OFFSET + 12
Private Const C_ERR_UNABLE_TO_CREATE_VALUE As Long = vbObjectError + C_ERR_OFFSET + 13
Private Const C_ERR_UNABLE_TO_DELETE_KEY As Long = vbObjectError + C_ERR_OFFSET + 14
Private Const C_ERR_UNABLE_TO_DELETE_VALUE As Long = vbObjectError + C_ERR_OFFSET + 15
Private Const C_ERR_INVALID_PATH As Long = vbObjectError + C_ERR_OFFSET + 16
Public Enum HKEY
    HKEY_CURRENT_USER_HKCU = &H80000001
    HKEY_LOCAL_MACHINE_HKLM = &H80000002
    HKEY_CLASSES_ROOT_HKCR = &H80000000
    HKEY_CURRENT_CONFIG_HKCC = &H80000005
    HKEY_DYN_DATA_HKDD = &H80000006
    HKEY_PERFORMANCE_DATA_HKPD = &H80000004
    HKEY_USERS_HKU = &H80000003
End Enum
Private Const KEY_QUERY_VALUE As Long = &H1
Private Const KEY_SET_VALUE As Long = &H2
Private Const KEY_CREATE_SUB_KEY As Long = &H4
Private Const KEY_ENUMERATE_SUB_KEYS As Long = &H8
Private Const KEY_NOTIFY As Long = &H10
Private Const KEY_CREATE_LINK As Long = &H20
Private Const KEY_ALL_ACCESS As Long = &H3F
Private Const REG_CREATED_NEW_KEY As Long = &H1
Private Const REG_OPENED_EXISTING_KEY As Long = &H2
Private Const STANDARD_RIGHTS_ALL As Long = &H1F0000
Private Const SPECIFIC_RIGHTS_ALL As Long = &HFFFF
Private Const REG_OPTION_NON_VOLATILE As Long = 0&
Private Const REG_OPTION_VOLATILE As Long = &H1
Private Const ERROR_SUCCESS As Long = 0&
Private Const ERROR_ACCESS_DENIED As Long = 5
Private Const ERROR_INVALID_DATA As Long = 13&
Private Const ERROR_MORE_DATA As Long = 234
Private Const ERROR_NO_MORE_ITEMS As Long = 259
Private Const S_OK   As Long = &H0
Private Const MAX_DATA_BUFFER_SIZE As Long = 1024
Private Const REGSTR_MAX_VALUE_LENGTH As Long = &H100
Private Type SECURITY_ATTRIBUTES
    nLength         As Long
    lpSecurityDescriptor As Long
    bInheritHandle  As Boolean
End Type
Private Type FILETIME
    dwLowDateTime   As Long
    dwHighDateTime  As Long
End Type
Public Enum REG_DATA_TYPE
    REG_INVALID = -1
    REG_SZ = 1
    REG_EXPAND_SZ = 2
```

```
        REG_BINARY = 3
        REG_DWORD = 4
        REG_MULTI_SZ = 7
    End Enum
    Private Type ACL
        AclRevision     As Byte
        Sbz1            As Byte
        AclSize         As Integer
        AceCount        As Integer
        Sbz2            As Integer
    End Type
    Private Type SECURITY_DESCRIPTOR
        Revision        As Byte
        Sbz1            As Byte
        control         As Long
        Owner           As Long
        Group           As Long
        Sacl            As ACL
        Dacl            As ACL
    End Type
    Private Declare PtrSafe Function RegCloseKey Lib "advapi32.dll" _
            (ByVal hiveKey As Long) As Long
    Private Declare PtrSafe Function RegCreateKeyEx Lib "advapi32.dll" Alias "RegCreateKeyExA" ( _                    RegistryEditor.
            ByVal hiveKey As Long, _
            ByVal lpSubKey As String, _
            ByVal Reserved As Long, _
            ByVal lpClass As String, _
            ByVal dwOptions As Long, _
            ByVal samDesired As Long, _
            lpSecurityAttributes As SECURITY_ATTRIBUTES, _
            phkResult As Long, _
            lpdwDisposition As Long) As Long
    Private Declare PtrSafe Function RegDeleteKey Lib "advapi32.dll" Alias "RegDeleteKeyA" ( _                        RegistryEditor.
            ByVal hiveKey As Long, _
            ByVal lpSubKey As String) As Long
    Private Declare PtrSafe Function RegOpenKey Lib "advapi32.dll" Alias "RegOpenKeyA" ( _                            RegistryEditor.
            ByVal hiveKey As Long, _
            ByVal lpSubKey As String, _
            phkResult As Long) As Long
    Private Declare PtrSafe Function RegDeleteValue Lib "advapi32.dll" Alias "RegDeleteValueA" ( _                    RegistryEditor.
            ByVal hiveKey As Long, _
            ByVal lpValueName As String) As Long
    Private Declare PtrSafe Function RegEnumKey Lib "advapi32.dll" Alias "RegEnumKeyA" ( _                            RegistryEditor.
            ByVal hiveKey As Long, _
            ByVal dwIndex As Long, _
            ByVal lpName As String, _
            ByVal cbName As Long) As Long
    Private Declare PtrSafe Function RegEnumKeyEx Lib "advapi32.dll" Alias "RegEnumKeyExA" ( _                        RegistryEditor.
            ByVal hiveKey As Long, _
            ByVal dwIndex As Long, _
            ByVal lpName As String, _
            lpcbName As Long, _
            ByVal lpReserved As Long, _
            ByVal lpClass As String, _
            lpcbClass As Long, _
            lpftLastWriteTime As FILETIME) As Long
    Private Declare PtrSafe Function RegEnumValue Lib "advapi32.dll" Alias "RegEnumValueA" ( _                        RegistryEditor.
            ByVal hiveKey As Long, _
            ByVal dwIndex As Long, _
            ByVal lpValueName As String, _
            lpcbValueName As Long, _
            ByVal lpReserved As Long, _
            lpType As Long, _
            lpData As Byte, _
            lpcbData As Long) As Long
    Private Declare PtrSafe Function RegFlushKey Lib "advapi32.dll" ( _                                               RegistryEditor.
            ByVal hiveKey As Long) As Long
    Private Declare PtrSafe Function RegGetKeySecurity Lib "advapi32.dll" ( _                                         RegistryEditor.
            ByVal hiveKey As Long, _
            ByVal SecurityInformation As Long, _
            pSecurityDescriptor As SECURITY_DESCRIPTOR, _
            lpcbSecurityDescriptor As Long) As Long
```

```
Private Declare PtrSafe Function RegQueryInfoKey Lib "advapi32.dll" Alias "RegQueryInfoKeyA" ( _                RegistryEditor.
        ByVal hiveKey As Long, _
        ByVal lpClass As String, _
        lpcbClass As Long, _
        ByVal lpReserved As Long, _
        lpcSubKeys As Long, _
        lpcbMaxSubKeyLen As Long, _
        lpcbMaxClassLen As Long, _
        lpcValues As Long, _
        lpcbMaxValueNameLen As Long, _
        lpcbMaxValueLen As Long, _
        lpcbSecurityDescriptor As Long, _
        lpftLastWriteTime As FILETIME) As Long
Private Declare PtrSafe Function RegQueryValue Lib "advapi32.dll" Alias "RegQueryValueA" ( _                    RegistryEditor.
        ByVal hiveKey As Long, _
        ByVal lpSubKey As String, _
        ByVal lpValue As String, _
        lpcbValue As Long) As Long
Private Declare PtrSafe Function RegQueryValueEx Lib "advapi32.dll" Alias "RegQueryValueExA" ( _                RegistryEditor.
        ByVal hiveKey As Long, _
        ByVal lpValueName As String, _
        ByVal lpReserved As Long, _
        lpType As Long, _
        lpData As Any, _
        lpcbData As Long) As Long
Private Declare PtrSafe Function RegSetValueEx Lib "advapi32.dll" Alias "RegSetValueExA" ( _                    RegistryEditor.
        ByVal hiveKey As Long, _
        ByVal lpValueName As String, _
        ByVal Reserved As Long, _
        ByVal dwType As Long, _
        lpData As Any, _
        ByVal cbData As Long) As Long
Private Declare PtrSafe Function RegSetValueExStr Lib "advapi32" Alias "RegSetValueExA" ( _                     RegistryEditor.
        ByVal hiveKey As Long, _
        ByVal lpValueName As String, _
        ByVal Reserved As Long, _
        ByVal dwType As Long, _
        ByVal szData As String, _
        ByVal cbData As Long) As Long
Private Declare PtrSafe Function RegSetValueExLong Lib "advapi32" Alias "RegSetValueExA" ( _                    RegistryEditor.
        ByVal hiveKey As Long, _
        ByVal lpValueName As String, _
        ByVal Reserved As Long, _
        ByVal dwType As Long, _
        szData As Long, _
        ByVal cbData As Long) As Long
Private Declare PtrSafe Function RegOpenKeyEx Lib "advapi32" Alias "RegOpenKeyExA" ( _                          RegistryEditor.
        ByVal hiveKey As Long, _
        ByVal lpSubKey As String, _
        ByVal ulOptions As Long, _
        ByVal samDesired As Long, _
        phkResult As Long) As Long
Private Declare PtrSafe Function RegQueryValueExStr Lib "advapi32" Alias "RegQueryValueExA" ( _                 RegistryEditor.
        ByVal hiveKey As Long, _
        ByVal lpValueName As String, _
        ByVal lpReserved As Long, _
        ByRef lpType As Long, _
        ByVal szData As String, _
        ByRef lpcbData As Long) As Long
Private Const VbaSettingsBasekey = "HKEY_CURRENT_USER\SOFTWARE\VB and VBA Program Settings"
Sub VBA_OpenSettings()                                                                                          ...
    Dim baseKey      As String
    baseKey = VbaSettingsBasekey
    Dim wsh          As Object
    ResetErrorVariables
    If Not RegistryUpdateValue(HKEY.HKEY_CURRENT_USER_HKCU,  _
    "Software\Microsoft\Windows\CurrentVersion\Applets\Regedit", _
            "LastKey", "Computer\" & baseKey, createKeyIfNotExist:=False) Then
        m_AppErr.Number = C_ERR_INVALID_PATH
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        GoTo ErrHandler
    End If
    CloseRegEdit promptUserBeforeClosing:=False
    On Error Resume Next
    Set wsh = VBA.CreateObject("WScript.Shell")
    wsh.Run "regedit.exe -m", 1, False
    Set wsh = Nothing
    On Error GoTo 0
    Exit Sub
ErrHandler:
    m_AppErr.Source = "OpenRegEditToKey(...)"
    OpenRegEdit openToLastKey:=False
    m_AppErr.DisplayMessage
End Sub
Function VBA_SaveSetting( _                                                                    ...
        sAPPNAME As String, _
        sSectionName As String, _
        sKeyName As String, _
        sSettingValue As String) As Boolean
    On Error GoTo Error_Handler
    Call SaveSetting(sAPPNAME, sSectionName, sKeyName, sSettingValue)
    VBA_SaveSetting = True
Error_Handler_Exit:
    On Error Resume Next
    Exit Function
Error_Handler:
    MsgBox "The following error has occurred" & vbCrLf & vbCrLf & _
            "Error Source: SaveRegistrySetting" & vbCrLf & _
            "Error Number: " & Err.Number & vbCrLf & _
            "Error Description: " & Err.Description & _
            Switch(Erl = 0, "", Erl <> 0, vbCrLf & "Line No: " & Erl) _
            , vbOKOnly + vbCritical, "An Error has Occurred!"
    Resume Error_Handler_Exit
End Function
Function VBA_GetSetting(sAPPNAME As String, sSectionName As String, sKeyName As String) As String  ...
    On Error GoTo Error_Handler
    VBA_GetSetting = GetSetting(sAPPNAME, sSectionName, sKeyName)
Error_Handler_Exit:
    On Error Resume Next
    Exit Function
Error_Handler:
    MsgBox "The following error has occurred" & vbCrLf & vbCrLf & _
            "Error Source: GetRegistrySetting" & vbCrLf & _
            "Error Number: " & Err.Number & vbCrLf & _
            "Error Description: " & Err.Description & _
            Switch(Erl = 0, "", Erl <> 0, vbCrLf & "Line No: " & Erl) _
            , vbOKOnly + vbCritical, "An Error has Occurred!"
    Resume Error_Handler_Exit
End Function
Function VBA_GetAllSettings(sAPPNAME As String, sSectionName As String) As Variant             ...
    On Error GoTo Error_Handler
    Dim aSectionSettings As Variant
    Dim iCounter    As Long
    aSectionSettings = GetAllSettings(sAPPNAME, sSectionName)
    VBA_GetAllSettings = aSectionSettings
Error_Handler_Exit:
    On Error Resume Next
    Exit Function
Error_Handler:
    MsgBox "The following error has occurred" & vbCrLf & vbCrLf & _
            "Error Source: GetAllRegistrySettings" & vbCrLf & _
            "Error Number: " & Err.Number & vbCrLf & _
            "Error Description: " & Err.Description & _
            Switch(Erl = 0, "", Erl <> 0, vbCrLf & "Line No: " & Erl) _
            , vbOKOnly + vbCritical, "An Error has Occurred!"
    Resume Error_Handler_Exit
End Function
Function VBA_DeleteSetting(sAPPNAME As String, sSectionName As String, Optional sKeyName As String)  _  ...
As Boolean
    On Error GoTo Error_Handler
    If sKeyName = "" Then
        Call DeleteSetting(sAPPNAME, sSectionName)
    Else
        Call DeleteSetting(sAPPNAME, sSectionName, sKeyName)
    End If
    VBA_DeleteSetting = True
Error_Handler_Exit:
    On Error Resume Next
    Exit Function
Error_Handler:
    If Err.Number <> 5 Then
        MsgBox "The following error has occurred" & vbCrLf & vbCrLf & _
                "Error Source: DeleteRegistrySetting" & vbCrLf & _
                "Error Number: " & Err.Number & vbCrLf & _
                "Error Description: " & Err.Description & _
                Switch(Erl = 0, "", Erl <> 0, vbCrLf & "Line No: " & Erl) _
                , vbOKOnly + vbCritical, "An Error has Occurred!"
    End If
    Resume Error_Handler_Exit
End Function
Public Property Get About() As String                                                                    ...
    About = "ChE Junkie VBA Registry class module, v" & Me.version & "." & VBA.vbCrLf
    About = About & "An extension of original work done by Chip Pearson (www.cpearson.com)." & VBA. _
    vbCrLf & VBA.vbCrLf
    About = About & "For additional details see:" & VBA.vbCrLf & "https://chejunkie. _
    com/knowledge-base/registry-editor-class-vba/"
End Property
Public Property Get AppErr() As ApplicationError                                                         ...
    Set AppErr = m_AppErr
End Property
Public Function GetBaseKeyName(baseKey As HKEY) As String                                                ...
    Select Case baseKey
        Case HKEY.HKEY_CLASSES_ROOT_HKCR: GetBaseKeyName = "HKEY_CLASSES_ROOT"
        Case HKEY.HKEY_CURRENT_USER_HKCU: GetBaseKeyName = "HKEY_CURRENT_USER"
        Case HKEY.HKEY_LOCAL_MACHINE_HKLM: GetBaseKeyName = "HKEY_LOCAL_MACHINE"
        Case HKEY.HKEY_USERS_HKU: GetBaseKeyName = "HKEY_USERS"
        Case HKEY.HKEY_CURRENT_CONFIG_HKCC: GetBaseKeyName = "HKEY_CURRENT_CONFIG"
        Case HKEY.HKEY_DYN_DATA_HKDD: GetBaseKeyName = "HKEY_DYN_DATA"
        Case HKEY.HKEY_PERFORMANCE_DATA_HKPD: GetBaseKeyName = "HKEY_PERFORMANCE_DATA"
    End Select
End Function
Public Function GetBaseKeyNameShort(baseKey As HKEY) As String                                           ...
    Select Case baseKey
        Case HKEY.HKEY_CLASSES_ROOT_HKCR: GetBaseKeyNameShort = "HKCR"
        Case HKEY.HKEY_CURRENT_USER_HKCU: GetBaseKeyNameShort = "HKCU"
        Case HKEY.HKEY_LOCAL_MACHINE_HKLM: GetBaseKeyNameShort = "HKLM"
        Case HKEY.HKEY_USERS_HKU: GetBaseKeyNameShort = "HKU"
        Case HKEY.HKEY_CURRENT_CONFIG_HKCC: GetBaseKeyNameShort = "HKCC"
        Case HKEY.HKEY_DYN_DATA_HKDD: GetBaseKeyNameShort = "HKDD"
        Case HKEY.HKEY_PERFORMANCE_DATA_HKPD: GetBaseKeyNameShort = "HKPD"
    End Select
End Function
Public Sub CloseRegEdit(Optional promptUserBeforeClosing As Boolean = True)                              ...
    Dim cReg         As Collection
    Dim proc         As Object
    Dim errReturnCode As Long
    Dim response     As Integer
    If IsRegEditOpen(cReg) Then
        If promptUserBeforeClosing Then
            Select Case cReg.Count
                Case Is = 1: response = VBA.MsgBox("Are you sure that you want to close the  _
                Regestry Editor (regedit.exe)?", vbYesNo)
                Case Is > 1: response = VBA.MsgBox("Are you sure that you want to close [" & cReg. _
                Count & "] instances of the Registry Editor (regedit.exe)?", vbYesNo)
            End Select
        End If
        If (response = vbYes) Or (promptUserBeforeClosing = False) Then
            For Each proc In cReg
                errReturnCode = proc.Terminate()
            Next proc
        End If
    End If
End Sub
Public Function IsRegEditOpen(Optional cReg As Collection) As Boolean                                    ...
    Dim oServ        As Object
    Dim cProc        As Variant
```

```vb
    Dim oProc       As Object
    Set oServ = GetObject("winmgmts:")
    Set cProc = oServ.execquery("Select * from Win32_Process")
    Set cReg = New Collection
    For Each oProc In cProc
        If (oProc.Name = "regedit.exe") Then
            cReg.Add oProc
        End If
    Next
    On Error GoTo ErrHandler
    If (cReg.Count > 0) Then
        IsRegEditOpen = True
    End If
ErrHandler:
End Function
Public Property Get Name() As String                                                                    ...
    Name = C_NAME
End Property
Public Sub OpenRegEdit(Optional openToLastKey As Boolean = True, Optional closeBeforeOpening =  _      ...
False)
    Dim wsh         As Object
    If Not openToLastKey Then
        RegistryUpdateValue HKEY.HKEY_CURRENT_USER_HKCU,  _
        "Software\Microsoft\Windows\CurrentVersion\Applets\Regedit", "LastKey", "Computer",  _
        createKeyIfNotExist:=False
    End If
    If closeBeforeOpening Then
        CloseRegEdit promptUserBeforeClosing:=False
    End If
    On Error Resume Next
    Set wsh = VBA.CreateObject("WScript.Shell")
    wsh.Run "regedit.exe -m", 1, False
    Set wsh = Nothing
    On Error GoTo 0
End Sub
Public Sub OpenRegEditToKey(baseKey As HKEY, ByVal Keyname As String, Optional closeBeforeOpening  _   ...
As Boolean = False)
    Dim wsh         As Object
    ResetErrorVariables
    If IsValidBaseKey(baseKey:=baseKey) = False Then
        m_AppErr.Number = C_ERR_INVALID_BASE_KEY
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        GoTo ErrHandler
    End If
    If IsValidKeyName(Keyname:=Keyname) = False Then
        m_AppErr.Number = C_ERR_INVALID_BASE_KEY
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        GoTo ErrHandler
    End If
    If RegistryKeyExists(baseKey:=baseKey, Keyname:=Keyname) = False Then
        m_AppErr.Number = C_ERR_KEY_NOT_FOUND
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        GoTo ErrHandler
    End If
    If Not RegistryUpdateValue(HKEY.HKEY_CURRENT_USER_HKCU,  _
    "Software\Microsoft\Windows\CurrentVersion\Applets\Regedit", _
            "LastKey", "Computer\" & GetBaseKeyName(baseKey) & "\" & Keyname, createKeyIfNotExist:= _
            False) Then
        m_AppErr.Number = C_ERR_INVALID_PATH
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        GoTo ErrHandler
    End If
    If closeBeforeOpening Then
        CloseRegEdit promptUserBeforeClosing:=False
    End If
    On Error Resume Next
    Set wsh = VBA.CreateObject("WScript.Shell")
    wsh.Run "regedit.exe -m", 1, False
    Set wsh = Nothing
    On Error GoTo 0
    Exit Sub
ErrHandler:
    m_AppErr.Source = "OpenRegEditToKey(...)"
    OpenRegEdit openToLastKey:=False
    m_AppErr.DisplayMessage
End Sub
Public Function RegistryGetValue(baseKey As HKEY, ByVal Keyname As String, valueName As String) As  _                    ...
Variant
    Dim hiveKey      As Long
    Dim Result       As Long
    Dim regDataType As REG_DATA_TYPE
    Dim lenData      As Long
    Dim longData     As Long
    Dim stringData  As String
    Dim intArr(0 To 1024) As Integer
    Dim lenStringData As Long
    ResetErrorVariables
    If (IsValidBaseKey(baseKey:=baseKey) = False) Then
        m_AppErr.Number = C_ERR_INVALID_BASE_KEY
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        m_AppErr.Source = "RegistryGetValue(...) As Variant"
        RegistryGetValue = Null
        Exit Function
    End If
    If (IsValidKeyName(Keyname:=Keyname) = False) Then
        m_AppErr.Number = C_ERR_INVALID_BASE_KEY
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        m_AppErr.Source = "RegistryGetValue(...) As Variant"
        RegistryGetValue = Null
        Exit Function
    End If
    If (RegistryKeyExists(baseKey:=baseKey, Keyname:=Keyname) = False) Then
        m_AppErr.Number = C_ERR_KEY_NOT_FOUND
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        m_AppErr.Source = "RegistryGetValue(...) As Variant"
        RegistryGetValue = Null
        Exit Function
    End If
    regDataType = RegistryGetValueType(baseKey:=baseKey, Keyname:=Keyname, valueName:=valueName)
    hiveKey = OpenRegistryKey(baseKey:=baseKey, Keyname:=Keyname)
    If (hiveKey = 0) Then
        m_AppErr.NumberDLL = Result
        m_AppErr.Number = C_ERR_UNABLE_TO_OPEN_KEY
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        m_AppErr.Source = "RegistryGetValue(...) As Variant"
        RegistryGetValue = Null
        Exit Function
    End If
    If (regDataType = REG_DWORD) Or (regDataType = REG_BINARY) Then
        Result = RegQueryValueEx(hiveKey:=hiveKey, lpValueName:=valueName, lpReserved:=0&, _
                lpType:=regDataType, lpData:=longData, lpcbData:=Len(longData))
        If (Result = ERROR_SUCCESS) Then
            RegistryGetValue = longData
            Exit Function
        Else
            m_AppErr.NumberDLL = Result
            m_AppErr.Number = C_ERR_UNABLE_TO_READ_VALUE
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            m_AppErr.Source = "RegistryGetValue(...) As Variant"
            RegCloseKey hiveKey
            RegistryGetValue = Null
            Exit Function
        End If
    ElseIf (regDataType = REG_SZ) Or (regDataType = REG_EXPAND_SZ) Or (regDataType = REG_MULTI_SZ)  _
    Then
        stringData = VBA.String$(MAX_DATA_BUFFER_SIZE, vbNullChar)
        lenStringData = VBA.Len(stringData)
        Result = RegQueryValueExStr(hiveKey:=hiveKey, lpValueName:=valueName, lpReserved:=0&, _
                lpType:=regDataType, szData:=stringData, lpcbData:=lenStringData)
        If (Result <> ERROR_SUCCESS) Then
            m_AppErr.NumberDLL = Result
            m_AppErr.Number = C_ERR_UNABLE_TO_READ_VALUE
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            m_AppErr.Source = "RegistryGetValue(...) As Variant"
            RegCloseKey hiveKey
            RegistryGetValue = Null
            Exit Function
        End If
        stringData = TrimToNull(stringData)
        RegistryGetValue = stringData
    Else
        m_AppErr.Number = C_ERR_INVALID_DATA_TYPE
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        m_AppErr.Source = "RegistryGetValue(...) As Variant"
        RegistryGetValue = Null
    End If
End Function
Public Function RegistryKeyExists(baseKey As HKEY, ByVal Keyname As String, Optional  _                  ...
createKeyIfNotExist As Boolean = False) As Boolean
    Dim hiveKey     As Long
    Dim Result      As Long
    ResetErrorVariables
    If (IsValidBaseKey(baseKey:=baseKey) = False) Then
        m_AppErr.Number = C_ERR_INVALID_BASE_KEY
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        m_AppErr.Source = "RegistryKeyExists(...) As Boolean"
        RegistryKeyExists = False
    End If
    If (IsValidKeyName(Keyname:=Keyname) = False) Then
        m_AppErr.Number = C_ERR_INVALID_BASE_KEY
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        m_AppErr.Source = "RegistryKeyExists(...) As Boolean"
        RegistryKeyExists = False
    End If
    Result = RegOpenKey(hiveKey:=baseKey, lpSubKey:=Keyname, phkResult:=hiveKey)
    If (Result = ERROR_SUCCESS) Then
        RegistryKeyExists = True
    Else
        RegistryKeyExists = False
        If (createKeyIfNotExist = True) Then
            Result = RegistryCreateKey(baseKey:=baseKey, Keyname:=Keyname)
            RegistryKeyExists = CBool(Result)
        End If
    End If
    RegCloseKey hiveKey:=hiveKey
End Function
Public Function RegistryNumberOfSubKeys(baseKey As HKEY, ByVal Keyname As String, Optional  _            ...
listOfSubKeyNames As Variant) As Long
    listOfSubKeyNames = RegistrySubKeyNamesToArray(baseKey, Keyname)
    If VBA.IsNull(listOfSubKeyNames) Then
        RegistryNumberOfSubKeys = -1
    Else
        RegistryNumberOfSubKeys = UBound(listOfSubKeyNames)
        If LBound(listOfSubKeyNames) = 0 Then RegistryNumberOfSubKeys = RegistryNumberOfSubKeys + 1
    End If
End Function
Public Function RegistryNumberOfValues(baseKey As HKEY, ByVal Keyname As String, Optional  _             ...
listOfValueNames As Variant) As Long
    listOfValueNames = RegistryValueNamesToArray(baseKey, Keyname)
    If VBA.IsNull(listOfValueNames) Then
        RegistryNumberOfValues = -1
    Else
        RegistryNumberOfValues = UBound(listOfValueNames)
        If LBound(listOfValueNames) = 0 Then RegistryNumberOfValues = RegistryNumberOfValues + 1
    End If
End Function
Public Function GetDataTypeName(dataType As REG_DATA_TYPE) As String                                     ...
    Select Case dataType
        Case REG_INVALID: GetDataTypeName = "REG_INVALID"
        Case REG_SZ: GetDataTypeName = "REG_SZ"
        Case REG_EXPAND_SZ: GetDataTypeName = "REG_EXPAND_SZ"
        Case REG_BINARY: GetDataTypeName = "REG_BINARY"
        Case REG_DWORD: GetDataTypeName = "REG_DWORD"
        Case REG_MULTI_SZ: GetDataTypeName = "REG_MULTI_SZ"
    End Select
End Function
```

```
Public Function RegistryValueExists(baseKey As HKEY, ByVal Keyname As String, valueName As String, _        ...
        Optional createKeyIfNotExist As Boolean = False, Optional CreateType As REG_DATA_TYPE =  _
        REG_DWORD) As Boolean
    Dim hiveKey     As Long
    Dim Result      As Long
    ResetErrorVariables
    If (IsValidBaseKey(baseKey:=baseKey) = False) Then
        m_AppErr.Number = C_ERR_INVALID_BASE_KEY
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        m_AppErr.Source = "RegistryValueExists(...) As Boolean"
        RegistryValueExists = False
    End If
    If (IsValidKeyName(Keyname:=Keyname) = False) Then
        m_AppErr.Number = C_ERR_INVALID_BASE_KEY
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        m_AppErr.Source = "RegistryValueExists(...) As Boolean"
        RegistryValueExists = False
    End If
    hiveKey = OpenRegistryKey(baseKey:=baseKey, Keyname:=Keyname)
    If (hiveKey = 0) Then
        m_AppErr.Number = C_ERR_UNABLE_TO_OPEN_KEY
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        m_AppErr.Source = "RegistryValueExists(...) As Boolean"
        RegistryValueExists = False
    End If
    Result = RegQueryValueEx(hiveKey:=hiveKey, lpValueName:=valueName, lpReserved:=0&, lpType:=0&,  _
    lpData:=0&, lpcbData:=0&)
    If (Result = ERROR_SUCCESS) Or (Result = ERROR_MORE_DATA) Then
        RegistryValueExists = True
    Else
        If (createKeyIfNotExist = True) Then
            If (CreateType = REG_DWORD) Then
                Result = RegistryCreateValue(baseKey:=baseKey, Keyname:=Keyname, valueName:= _
                valueName, _
                        valueValue:=0&, createKeyIfNotExist:=True)
            Else
                Result = RegistryCreateValue(baseKey:=baseKey, Keyname:=Keyname, valueName:= _
                valueName, _
                        valueValue:=vbNullString, createKeyIfNotExist:=True)
            End If
            If (CBool(Result) = True) Then
                RegistryValueExists = True
            Else
                RegistryValueExists = False
            End If
        End If
    End If
    RegCloseKey hiveKey
End Function
Public Function RegistrySubKeyNamesToArray(baseKey As HKEY, ByVal Keyname As String) As Variant            ...
    Dim procHiveKeyRes As Long
    Dim Result      As Long
    Dim ooReg       As Object
    ResetErrorVariables
    If (IsValidBaseKey(baseKey:=baseKey) = False) Then
        m_AppErr.Number = C_ERR_INVALID_BASE_KEY
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        m_AppErr.Source = "RegistrySubKeyNames(...) As Variant"
        RegistrySubKeyNamesToArray = Null
        Exit Function
    End If
    If (IsValidKeyName(Keyname:=Keyname) = False) Then
        m_AppErr.Number = C_ERR_INVALID_BASE_KEY
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
        m_AppErr.Source = "RegistrySubKeyNames(...) As Variant"
        RegistrySubKeyNamesToArray = Null
        Exit Function
    End If
    procHiveKeyRes = OpenRegistryKey(baseKey:=baseKey, Keyname:=Keyname)
    If (procHiveKeyRes = 0) Then
        m_AppErr.Number = C_ERR_UNABLE_TO_OPEN_KEY
        m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
```

```
            m_AppErr.Source = "RegistrySubKeyNames(...) As Variant"
            RegistrySubKeyNamesToArray = Null
            RegCloseKey procHiveKeyRes
            Exit Function
        Else: RegCloseKey procHiveKeyRes
        End If
        On Error Resume Next
        Set ooReg = VBA.GetObject("winmgmts:{impersonationLevel=impersonate}!\\. _
        \root\default:StdRegProv")
        ooReg.EnumKey baseKey, Keyname, RegistrySubKeyNamesToArray
        If (Err.Number <> 0) Then
            m_AppErr.NumberDLL = Err.LastDllError
            m_AppErr.Number = Err.Number
            m_AppErr.Description = Err.Description
            m_AppErr.Source = "RegistrySubKeyNames(...) As Variant"
        End If
        Set ooReg = Nothing
        On Error GoTo 0
    End Function
    Public Function RegistryValueNamesToArray(baseKey As HKEY, ByVal Keyname As String) As Variant                    ...
        Dim procHiveKeyRes As Long
        Dim Result      As Long
        Dim ooReg       As Object
        ResetErrorVariables
        If (IsValidBaseKey(baseKey:=baseKey) = False) Then
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            m_AppErr.Source = "RegistryValueNamesToArray(...) As Variant"
            RegistryValueNamesToArray = Null
            Exit Function
        End If
        If (IsValidKeyName(Keyname:=Keyname) = False) Then
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            m_AppErr.Source = "RegistryValueNamesToArray(...) As Variant"
            RegistryValueNamesToArray = Null
            Exit Function
        End If
        procHiveKeyRes = OpenRegistryKey(baseKey:=baseKey, Keyname:=Keyname)
        If (procHiveKeyRes = 0) Then
            m_AppErr.Number = C_ERR_UNABLE_TO_OPEN_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            m_AppErr.Source = "RegistryValueNamesToArray(...) As Variant"
            RegistryValueNamesToArray = Null
            RegCloseKey procHiveKeyRes
            Exit Function
        Else: RegCloseKey procHiveKeyRes
        End If
        Set ooReg = VBA.GetObject("winmgmts:{impersonationLevel=impersonate}!\\. _
        \root\default:StdRegProv")
        On Error Resume Next
        ooReg.EnumValues baseKey, Keyname, RegistryValueNamesToArray
        If (Err.Number <> 0) Then
            m_AppErr.NumberDLL = Err.LastDllError
            m_AppErr.Number = Err.Number
            m_AppErr.Description = Err.Description
            m_AppErr.Source = "RegistryValueNamesToArray(...) As Variant"
        End If
        Set ooReg = Nothing
        On Error GoTo 0
    End Function
    Public Function RegistryGetValueType(baseKey As HKEY, ByVal Keyname As String, valueName As String)  _          ...
    As REG_DATA_TYPE
        Dim Result      As Long
        Dim procHiveKeyRes As Long
        Dim dataType    As REG_DATA_TYPE
        ResetErrorVariables
        If (IsValidBaseKey(baseKey:=baseKey) = False) Then
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            m_AppErr.Source = "RegistryGetValueType(...) As REG_DATA_TYPE"
            RegistryGetValueType = False
        End If
        If (IsValidKeyName(Keyname:=Keyname) = False) Then
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            m_AppErr.Source = "RegistryGetValueType(...) As REG_DATA_TYPE"
            RegistryGetValueType = False
        End If
        Result = RegOpenKey(hiveKey:=baseKey, lpSubKey:=Keyname, phkResult:=procHiveKeyRes)
        If (Result <> ERROR_SUCCESS) Then
            m_AppErr.Number = C_ERR_UNABLE_TO_OPEN_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            m_AppErr.Source = "RegistryGetValueType(...) As REG_DATA_TYPE"
            RegistryGetValueType = REG_INVALID
            Exit Function
        End If
        Result = RegQueryValueEx(hiveKey:=procHiveKeyRes, lpValueName:=valueName, lpReserved:=0&,  _
        lpType:=dataType, lpData:=0&, lpcbData:=0&)
        If (Result <> ERROR_SUCCESS) And (Result <> ERROR_MORE_DATA) Then
            m_AppErr.Number = C_ERR_UNABLE_TO_READ_VALUE
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            m_AppErr.Source = "RegistryGetValueType(...) As REG_DATA_TYPE"
            RegistryGetValueType = REG_INVALID
            RegCloseKey procHiveKeyRes
            Exit Function
        End If
        Select Case dataType
            Case REG_SZ
                RegistryGetValueType = REG_SZ
            Case REG_EXPAND_SZ
                RegistryGetValueType = REG_EXPAND_SZ
            Case REG_BINARY
                RegistryGetValueType = REG_BINARY
            Case REG_DWORD
                RegistryGetValueType = REG_DWORD
            Case REG_MULTI_SZ
                RegistryGetValueType = REG_MULTI_SZ
            Case Else
                RegistryGetValueType = REG_INVALID
        End Select
        RegCloseKey procHiveKeyRes
    End Function
    Public Function RegistryCreateValue(baseKey As HKEY, ByVal Keyname As String, valueName As String,  _                  ...
    valueValue As Variant, _
            Optional createKeyIfNotExist As Boolean = False) As Boolean
        Dim procHiveKeyRes As Long
        Dim Result      As Long
        Dim dataType    As REG_DATA_TYPE
        Dim StringValue As String
        Dim LongValue   As Long
        ResetErrorVariables
        If (IsValidBaseKey(baseKey:=baseKey) = False) Then
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            RegistryCreateValue = False
            Exit Function
        End If
        If (IsValidKeyName(Keyname:=Keyname) = False) Then
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            RegistryCreateValue = False
            Exit Function
        End If
        If (RegistryKeyExists(baseKey:=baseKey, Keyname:=Keyname, _
                createKeyIfNotExist:=createKeyIfNotExist) = False) Then
            m_AppErr.Number = C_ERR_KEY_NOT_FOUND
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            RegistryCreateValue = False
            Exit Function
        End If
        If (IsCompatibleValueValue(var:=valueValue) = False) Then
            m_AppErr.Number = C_ERR_INVALID_DATA_TYPE
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            RegistryCreateValue = False
            Exit Function
        End If
        If (RegistryKeyExists(baseKey:=baseKey, Keyname:=Keyname, createKeyIfNotExist:=False) = False)  _
        Then
            If (createKeyIfNotExist = True) Then
                If (RegistryKeyExists(baseKey:=baseKey, Keyname:=Keyname, createKeyIfNotExist:=True) =  _
                False) Then
                    m_AppErr.Number = C_ERR_UNABLE_TO_CREATE_KEY
                    m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
                    RegistryCreateValue = False
                    Exit Function
                End If
            Else
                m_AppErr.Number = C_ERR_KEY_NOT_FOUND
                m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
                RegistryCreateValue = False
                Exit Function
            End If
        End If
        If (RegistryValueExists(baseKey:=baseKey, Keyname:=Keyname, valueName:=valueName) = True) Then
            dataType = RegistryGetValueType(baseKey:=baseKey, Keyname:=Keyname, valueName:=valueName)
            If (dataType = REG_SZ) Then
                If (VarType(valueValue) <> vbString) Then
                    m_AppErr.Number = C_ERR_DATA_TYPE_MISMATCH
                    m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
                    RegistryCreateValue = False
                    Exit Function
                Else
                End If
            Else
            End If
        Else
            If (VarType(valueValue) = vbString) Then
                dataType = REG_SZ
            Else
                dataType = REG_DWORD
            End If
        End If
        If (dataType = REG_DWORD) Then
            LongValue = VBA.CLng(valueValue)
            procHiveKeyRes = OpenRegistryKey(baseKey:=baseKey, Keyname:=Keyname)
            If (procHiveKeyRes = 0) Then
                m_AppErr.NumberDLL = Err.LastDllError
                m_AppErr.Number = C_ERR_UNABLE_TO_OPEN_KEY
                m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
                RegCloseKey procHiveKeyRes
                RegistryCreateValue = False
                Exit Function
            End If
            Result = RegSetValueExLong(hiveKey:=procHiveKeyRes, lpValueName:=valueName, Reserved:=0&, _
                    dwType:=REG_DWORD, szData:=LongValue, cbData:=Len(LongValue))
            If (Result <> ERROR_SUCCESS) Then
                m_AppErr.Number = C_ERR_UNABLE_TO_UPDATE_VALUE
                m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
                RegCloseKey procHiveKeyRes
                RegistryCreateValue = False
                Exit Function
            End If
        Else
            StringValue = CStr(valueValue)
            procHiveKeyRes = OpenRegistryKey(baseKey:=baseKey, Keyname:=Keyname)
            If (procHiveKeyRes = 0) Then
                m_AppErr.NumberDLL = Err.LastDllError
                m_AppErr.Number = C_ERR_UNABLE_TO_OPEN_KEY
                m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
                RegCloseKey procHiveKeyRes
                RegistryCreateValue = False
                Exit Function
            End If
            Result = RegSetValueExStr(hiveKey:=procHiveKeyRes, lpValueName:=valueName, Reserved:=0&, _
                    dwType:=REG_SZ, szData:=StringValue, cbData:=Len(StringValue))
            If (Result <> ERROR_SUCCESS) Then
                m_AppErr.Number = C_ERR_UNABLE_TO_UPDATE_VALUE
                m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
                RegistryCreateValue = False
                RegCloseKey procHiveKeyRes
                Exit Function
            End If
        End If
        RegCloseKey procHiveKeyRes
        RegistryCreateValue = True
    End Function
    Public Function RegistryCreateKey(baseKey As HKEY, ByVal Keyname As String) As Boolean                    ...
        Dim Result      As Long
        Dim procHiveKeyRes As Long
        Dim dataType    As REG_DATA_TYPE
        Dim secAttrib   As SECURITY_ATTRIBUTES
        Dim disposition As Long
        ResetErrorVariables
        If (IsValidBaseKey(baseKey:=baseKey) = False) Then
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            RegistryCreateKey = False
        End If
        If (IsValidKeyName(Keyname:=Keyname) = False) Then
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            RegistryCreateKey = False
        End If
        If (RegistryKeyExists(baseKey:=baseKey, Keyname:=Keyname) = True) Then
            RegistryCreateKey = True
            Exit Function
        End If
        Result = RegCreateKeyEx(hiveKey:=baseKey, lpSubKey:=Keyname, Reserved:=0&, lpClass:= _
        vbNullString, _
                dwOptions:=REG_OPTION_NON_VOLATILE, samDesired:=KEY_ALL_ACCESS, _
                lpSecurityAttributes:=secAttrib, phkResult:=procHiveKeyRes, lpdwDisposition:= _
                disposition)
        If (Result <> ERROR_SUCCESS) Then
            m_AppErr.NumberDLL = Result
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            RegistryCreateKey = False
            Exit Function
        End If
        RegistryCreateKey = True
    End Function
    Public Function RegistryDeleteValue(baseKey As HKEY, ByVal Keyname As String, valueName As String)  _    ...
    As Boolean
        Dim Result      As Long
        Dim procHiveKeyRes As Long
        Dim dataType    As REG_DATA_TYPE
        Dim secAttrib   As SECURITY_ATTRIBUTES
        Dim disposition As Long
        ResetErrorVariables
        If (IsValidBaseKey(baseKey:=baseKey) = False) Then
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            RegistryDeleteValue = False
            Exit Function
        End If
        If (IsValidKeyName(Keyname:=Keyname) = False) Then
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            RegistryDeleteValue = False
            Exit Function
        End If
        If (RegistryKeyExists(baseKey:=baseKey, Keyname:=Keyname, createKeyIfNotExist:=False) = False)  _
        Then
            m_AppErr.Number = C_ERR_KEY_NOT_FOUND
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            RegistryDeleteValue = False
            Exit Function
        End If
        procHiveKeyRes = OpenRegistryKey(baseKey:=baseKey, Keyname:=Keyname)
        If (procHiveKeyRes = 0) Then
            RegistryDeleteValue = False
            Exit Function
        End If
        If RegistryValueExists(baseKey:=baseKey, Keyname:=Keyname, valueName:=valueName) = False Then
            RegCloseKey procHiveKeyRes
            RegistryDeleteValue = True
            Exit Function
        End If
        Result = RegDeleteValue(hiveKey:=procHiveKeyRes, lpValueName:=valueName)
        If (Result <> ERROR_SUCCESS) Then
            m_AppErr.Number = C_ERR_UNABLE_TO_DELETE_VALUE
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            RegCloseKey procHiveKeyRes
            RegistryDeleteValue = False
            Exit Function
        End If
        RegCloseKey procHiveKeyRes
        RegistryDeleteValue = True
    End Function
    Public Function RegistryDeleteKey(baseKey As HKEY, ByVal Keyname As String) As Boolean                    ...
        Dim Result      As Long
        Dim procHiveKeyRes As Long
        Dim dataType    As REG_DATA_TYPE
        Dim secAttrib   As SECURITY_ATTRIBUTES
        Dim disposition As Long
        ResetErrorVariables
        If (IsValidBaseKey(baseKey:=baseKey) = False) Then
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            RegistryDeleteKey = False
            Exit Function
        End If
        If (IsValidKeyName(Keyname:=Keyname) = False) Then
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            RegistryDeleteKey = False
            Exit Function
        End If
        If (RegistryKeyExists(baseKey:=baseKey, Keyname:=Keyname, createKeyIfNotExist:=False) = False)  _
        Then
            RegistryDeleteKey = True
            Exit Function
        End If
        procHiveKeyRes = OpenRegistryKey(baseKey:=baseKey, Keyname:=Keyname)
        If (procHiveKeyRes = 0) Then
            RegistryDeleteKey = False
            Exit Function
        End If
        Result = RegDeleteKey(hiveKey:=baseKey, lpSubKey:=Keyname)
        RegCloseKey procHiveKeyRes
        If (Result <> ERROR_SUCCESS) Then
            m_AppErr.NumberDLL = Result
            m_AppErr.Number = C_ERR_UNABLE_TO_DELETE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            RegistryDeleteKey = False
            Exit Function
        End If
        RegistryDeleteKey = True
    End Function
    Public Function RegistryUpdateValue(baseKey As HKEY, ByVal Keyname As String, valueName As String,  _    ...
    NewValue As Variant, _
            Optional createKeyIfNotExist As Boolean = True) As Boolean
        Dim Result      As Boolean
        Dim hiveKey     As Long
        ResetErrorVariables
        If (IsValidBaseKey(baseKey:=baseKey) = False) Then
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            m_AppErr.Source = "RegistryUpdateValue(...) As Variant"
            RegistryUpdateValue = False
            Exit Function
        End If
        If (IsValidKeyName(Keyname:=Keyname) = False) Then
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            m_AppErr.Source = "RegistryUpdateValue(...) As Variant"
            RegistryUpdateValue = False
            Exit Function
        End If
        If (IsCompatibleValueValue(var:=NewValue) = False) Then
            m_AppErr.Number = C_ERR_INVALID_DATA_TYPE
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            m_AppErr.Source = "RegistryUpdateValue(...) As Variant"
            RegistryUpdateValue = False
            Exit Function
        End If
        Result = RegistryKeyExists(baseKey:=baseKey, Keyname:=Keyname, createKeyIfNotExist:=True)
        If (Result = False) Then
            m_AppErr.Number = C_ERR_KEY_NOT_FOUND
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            m_AppErr.Source = "RegistryUpdateValue(...) As Variant"
            RegistryUpdateValue = False
            Exit Function
        End If
        If (VarType(NewValue) = vbString) Then
            Result = RegistryValueExists(baseKey:=baseKey, Keyname:=Keyname, valueName:=valueName, _
                    createKeyIfNotExist:=createKeyIfNotExist, CreateType:=REG_DWORD)
        Else
            Result = RegistryValueExists(baseKey:=baseKey, Keyname:=Keyname, valueName:=valueName, _
                    createKeyIfNotExist:=createKeyIfNotExist, CreateType:=REG_SZ)
        End If
        If (Result = False) Then
            m_AppErr.Number = C_ERR_VALUE_NOT_FOUND
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            m_AppErr.Source = "RegistryUpdateValue(...) As Variant"
            RegistryUpdateValue = False
            Exit Function
        End If
        Result = RegistryDeleteValue(baseKey:=baseKey, Keyname:=Keyname, valueName:=valueName)
        Result = RegistryCreateValue(baseKey:=baseKey, Keyname:=Keyname, valueName:=valueName,  _
        valueValue:=NewValue, createKeyIfNotExist:=True)
        RegistryUpdateValue = Result
    End Function
    Private Function OpenRegistryKey(baseKey As HKEY, ByVal Keyname As String) As Long                    ...
        Dim Result      As Long
        Dim procHiveKeyRes As Long
        ResetErrorVariables
        If (IsValidBaseKey(baseKey) = False) Then
            OpenRegistryKey = 0
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            Exit Function
        End If
        Result = RegOpenKeyEx(hiveKey:=baseKey, lpSubKey:=Keyname, ulOptions:=0&, samDesired:= _
        KEY_ALL_ACCESS, phkResult:=procHiveKeyRes)
        If (Result <> ERROR_SUCCESS) Then
            OpenRegistryKey = 0
            m_AppErr.NumberDLL = Result
            m_AppErr.Number = C_ERR_INVALID_BASE_KEY
            m_AppErr.Description = GetAppErrDescription(m_AppErr.Number)
            Exit Function
        End If
        OpenRegistryKey = procHiveKeyRes
    End Function
    Private Function TrimToNull(TEXT As String, Optional Reverse As Boolean = False) As String            ...
        Dim pos         As Long
        If (Reverse = False) Then
            pos = VBA.InStr(1, TEXT, vbNullChar, vbTextCompare)
        Else
            pos = VBA.InStrRev(TEXT, vbNullChar, -1, vbTextCompare)
        End If
```

```
        If pos Then
            TrimToNull = VBA.Left(TEXT, pos - 1)
        Else
            TrimToNull = TEXT
        End If
    End Function
    Private Function TrimToChar(TEXT As String, Char As String, Optional ByVal Reverse As Boolean =  _                     ...
    False, _
            Optional ByVal CompaRemode As VbCompareMethod) As String
        Dim pos         As Long
        If (CompaRemode <> vbBinaryCompare) Then
            CompaRemode = vbTextCompare
        End If
        If (Reverse = False) Then
            pos = InStr(1, TEXT, Char, CompaRemode)
        Else
            pos = InStrRev(TEXT, Char, -1, CompaRemode)
        End If
        If pos Then
            TrimToChar = VBA.Left(TEXT, pos - 1)
        Else
            TrimToChar = TEXT
        End If
    End Function
    Private Function IsValidBaseKey(baseKey As HKEY) As Boolean                                                           ...
        Select Case baseKey
            Case HKEY.HKEY_CURRENT_USER_HKCU, HKEY.HKEY_LOCAL_MACHINE_HKLM, _
                    HKEY.HKEY_CLASSES_ROOT_HKCR, HKEY.HKEY_CURRENT_CONFIG_HKCC, HKEY.HKEY_DYN_DATA_HKDD, _
                    _
                    HKEY.HKEY_PERFORMANCE_DATA_HKPD, HKEY.HKEY_USERS_HKU
                IsValidBaseKey = True
            Case Else
                IsValidBaseKey = False
        End Select
    End Function
    Private Sub ResetErrorVariables()                                                                                     ...
        m_AppErr.clear
    End Sub
    Private Function GetAppErrDescription(errNumber As Long) As String                                                    ...
        Select Case errNumber
            Case C_ERR_NO_ERROR: GetAppErrDescription = vbNullString
            Case C_ERR_INVALID_BASE_KEY: GetAppErrDescription = "Invalid Base Key Value."
            Case C_ERR_INVALID_DATA_TYPE: GetAppErrDescription = "Invalid Data Type."
            Case C_ERR_KEY_NOT_FOUND: GetAppErrDescription = "Key Not Found."
            Case C_ERR_VALUE_NOT_FOUND: GetAppErrDescription = "Value Not Found."
            Case C_ERR_DATA_TYPE_MISMATCH: GetAppErrDescription = "Value Data Type Mismatch."
            Case C_ERR_ENTRY_LOCKED: GetAppErrDescription = "Registry Entry Locked."
            Case C_ERR_INVALID_KEYNAME: GetAppErrDescription = "The Specified Key Is Invalid."
            Case C_ERR_UNABLE_TO_OPEN_KEY: GetAppErrDescription = "Unable To Open Key."
            Case C_ERR_UNABLE_TO_READ_KEY: GetAppErrDescription = "Unable To Read Key."
            Case C_ERR_UNABLE_TO_CREATE_KEY: GetAppErrDescription = "Unable To Create Key."
            Case C_ERR_UNABLE_TO_READ_VALUE: GetAppErrDescription = "Unable To Read Value."
            Case C_ERR_UNABLE_TO_UPDATE_VALUE: GetAppErrDescription = "Unable To Update Value."
            Case C_ERR_UNABLE_TO_CREATE_VALUE: GetAppErrDescription = "Unable To Create Value."
            Case C_ERR_UNABLE_TO_DELETE_KEY: GetAppErrDescription = "Unable To Delete Key."
            Case C_ERR_UNABLE_TO_DELETE_VALUE: GetAppErrDescription = "Unable To Delete Value."
            Case C_ERR_INVALID_PATH: GetAppErrDescription = "Invalid registry path."
            Case Else
                GetAppErrDescription = "Undefined Error."
        End Select
    End Function
    Private Function IsStringValidLength(txt As String) As Boolean                                                        ...
        IsStringValidLength = (Len(txt) <= REGSTR_MAX_VALUE_LENGTH)
    End Function
    Private Function IsValidKeyName(Keyname As String) As Boolean                                                         ...
        IsValidKeyName = (VBA.Len(Keyname) <= REGSTR_MAX_VALUE_LENGTH) And (Len(VBA.Trim(Keyname)) > 0)
        If (VBA.Mid(Keyname, 1, 1) = "\") Then
            Do While VBA.Mid(Keyname, 1, 1) = "\"
                Keyname = VBA.Mid(Keyname, 2, VBA.Len(Keyname) - 1)
            Loop
        End If
        If (VBA.Mid(Keyname, VBA.Len(Keyname), 1) = "\") Then
```

```
            Do While VBA.Mid(Keyname, VBA.Len(Keyname), 1) = "\"
                Keyname = VBA.Mid(Keyname, 1, VBA.Len(Keyname) - 1)
            Loop
        End If
End Function
Private Function IsValidDataType(dataType As REG_DATA_TYPE) As Boolean                                                                ...
    Select Case dataType
        Case REG_SZ, REG_DWORD
            IsValidDataType = True
        Case Else
            IsValidDataType = False
    End Select
End Function
Private Function IsCompatibleValueValue(var As Variant) As Boolean                                                                  ...
    If VarType(var) >= vbArray Then
        IsCompatibleValueValue = False
        Exit Function
    End If
    If IsArray(var) = True Then
        IsCompatibleValueValue = False
        Exit Function
    End If
    If IsObject(var) = True Then
        IsCompatibleValueValue = False
        Exit Function
    End If
    Select Case VarType(var)
        Case vbBoolean, vbByte, vbCurrency, vbDate, vbDouble, vbInteger, vbLong, vbSingle, vbString
            IsCompatibleValueValue = True
        Case Else
            IsCompatibleValueValue = False
    End Select
End Function
Public Property Get version() As String                                                                                              ...
    version = "3.0 (2017)"
End Property
Private Sub Class_Initialize()                                                                                                       ...
    Debug.Print "|* Initializing Class:= " & C_NAME
    Set m_AppErr = New ApplicationError
    m_AppErr.Initialize C_NAME
End Sub
```

--- ApplicationError ---

```
Option Explicit
Option Base 1
Private Const C_NAME As String = "ApplicationError.cls"
Private Const C_ERR_NO_ERROR As Long = 0
Private m_ErrNumber As Long
Private m_ErrNumberDLL As Long
Private m_ErrSource As String
Private m_ErrDescription As String
Private m_ErrDescriptionDLL As String
Private m_ParentName As String
Private m_HasError  As Boolean
Private Const FORMAT_MESSAGE_ALLOCATE_BUFFER As Long = &H100
Private Const FORMAT_MESSAGE_ARGUMENT_ARRAY As Long = &H2000
Private Const FORMAT_MESSAGE_FROM_HMODULE As Long = &H800
Private Const FORMAT_MESSAGE_FROM_STRING As Long = &H400
Private Const FORMAT_MESSAGE_FROM_SYSTEM As Long = &H1000
Private Const FORMAT_MESSAGE_MAX_WIDTH_MASK As Long = &HFF
Private Const FORMAT_MESSAGE_IGNORE_INSERTS As Long = &H200
Private Const FORMAT_MESSAGE_TEXT_LEN As Long = &HA0
Private Declare PtrSafe Function FormatMessage Lib "kernel32" _
        Alias "FormatMessageA" ( _
        ByVal dwFlags As Long, _
        ByVal lpSource As Any, _
        ByVal dwMessageId As Long, _
        ByVal dwLanguageId As Long, _
        ByVal lpBuffer As String, _
        ByVal nSize As Long, _
        ByRef Arguments As Long) As Long
Private Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)          ApplicationError.
Public Sub clear()
    m_ErrNumber = C_ERR_NO_ERROR
    m_ErrNumberDLL = C_ERR_NO_ERROR
    m_ErrDescription = ""
    m_ErrSource = ""
    m_HasError = False
End Sub
Public Sub Define(errNumber As Long, errDescription As String, Optional errSource As String)
    clear
    Me.Number = errNumber
    m_ErrDescription = errDescription
    m_ErrSource = errSource
End Sub
Public Property Get Description() As String
    Description = m_ErrDescription
End Property
Property Let Description(str As String)
    m_ErrDescription = str
End Property
Public Sub DisplayMessage(Optional displayTitle As String, Optional appendMessage As String,  _
Optional msgBoxStyle As VbMsgBoxStyle = vbExclamation)
    If (displayTitle = "") Then
        displayTitle = "--ERROR!--"
    End If
    If (appendMessage <> vbNullString) Then
        MsgBox GetDescription & VBA.vbCrLf & VBA.vbCrLf & appendMessage, Title:=displayTitle,  _
        Buttons:=msgBoxStyle
    Else
        MsgBox GetDescription, Title:=displayTitle, Buttons:=msgBoxStyle
    End If
End Sub
Private Function GetDescription() As String
    If (m_ParentName <> "") Then
        GetDescription = "|> Parent: " & m_ParentName & VBA.vbCrLf
    End If
    If (m_ErrSource <> "") Then
        GetDescription = GetDescription & _
                "|> Source: " & m_ErrSource & VBA.vbCrLf & _
                "|> Number: " & m_ErrNumber & VBA.vbCrLf & _
                "|> " & m_ErrDescription
    Else
```

```
        GetDescription = GetDescription & _
                "|> Number: " & m_ErrNumber & VBA.vbCrLf & _
                "|> " & m_ErrDescription
    End If
    If (m_ErrDescriptionDLL <> "") Then
        GetDescription = GetDescription & VBA.vbCrLf & VBA.vbCrLf & _
                String(50, "-") & VBA.vbCrLf & _
                "|> DLL error number: " & m_ErrNumberDLL & VBA.vbCrLf & _
                "|> " & m_ErrDescription
    End If
End Function
Public Property Get About() As String                                                          ...
    About = "ChE Junkie VBA Application Error class module, Version " & Me.version & "." & VBA. _
    vbCrLf & VBA.vbCrLf
    About = About & "For additional details see:" & VBA.vbCrLf & "https://chejunkie. _
    com/knowledge-base/application-error-class-vba"
End Property
Public Property Get Name() As String                                                           ...
    Name = C_NAME
End Property
Public Property Get version() As String                                                        ...
    version = "Version 1.0 (2017)"
End Property
Property Get HasError() As Boolean                                                             ...
    HasError = m_HasError
End Property
Public Sub Initialize(parentName_ As String)                                                   ...
    clear
    m_ParentName = parentName_
End Sub
Public Property Get Number() As Long                                                           ...
    Number = m_ErrNumber
End Property
Public Property Get NumberDLL() As Long                                                        ...
    NumberDLL = m_ErrNumberDLL
End Property
Private Function GetSystemErrorMessageText(errNumber As Long) As String                        ...
    Dim ErrorText   As String
    Dim textLen     As Long
    Dim FormatMessageResult As Long
    Dim langID      As Long
    langID = 0&
    ErrorText = VBA.String$(FORMAT_MESSAGE_TEXT_LEN, vbNullChar)
    textLen = FORMAT_MESSAGE_TEXT_LEN
    FormatMessageResult = FormatMessage( _
            dwFlags:=FORMAT_MESSAGE_FROM_SYSTEM Or _
            FORMAT_MESSAGE_IGNORE_INSERTS, _
            lpSource:=0&, _
            dwMessageId:=errNumber, _
            dwLanguageId:=langID, _
            lpBuffer:=ErrorText, _
            nSize:=textLen, _
            Arguments:=0&)
    If FormatMessageResult = 0& Then
        MsgBox "An error occurred with the FormatMessage" & _
                " API function call." & vbCrLf & _
                "Error: " & CStr(Err.LastDllError) & _
                " VBA.Hex(" & VBA.Hex(Err.LastDllError) & ")."
        GetSystemErrorMessageText = "An internal system error occurred with the" & vbCrLf & _
                "FormatMessage API function: " & CStr(Err.LastDllError) & ". No futher information"  _
                & vbCrLf & _
                "is available."
        Exit Function
    End If
    ErrorText = VBA.Left$(ErrorText, FormatMessageResult)
    If VBA.Len(ErrorText) >= 2 Then
        If VBA.Right$(ErrorText, 2) = vbCrLf Then
            ErrorText = VBA.Left$(ErrorText, VBA.Len(ErrorText) - 2)
        End If
    End If
    GetSystemErrorMessageText = ErrorText
End Function
```

```
Property Let Number(lng As Long)                                                              ...
    If (lng <> C_ERR_NO_ERROR) Then
        m_ErrNumber = lng
        m_HasError = True
    End If
End Property
Property Let NumberDLL(lng As Long)                                                           ...
    If (lng <> C_ERR_NO_ERROR) Then
        m_ErrNumberDLL = lng
        m_ErrDescriptionDLL = GetSystemErrorMessageText(lng)
        m_HasError = True
    End If
End Property
Private Property Get ParentName() As String                                                   ...
    ParentName = m_ParentName
End Property
Public Sub PrintMessage()                                                                     ...
    Debug.Print ""
    Debug.Print String(50, "=")
    Debug.Print " Application Error"
    Debug.Print String(50, "=")
    Debug.Print GetDescription
End Sub
Public Property Get Source() As String                                                        ...
    Source = m_ErrSource
End Property
Property Let Source(str As String)                                                            ...
    m_ErrSource = str
End Property
Private Sub Class_Initialize()                                                                ...
    Debug.Print "|* Initializing Class:= " & C_NAME
End Sub
```

--- BetterArray ---

```
Option Explicit
Private Const DEFAULT_CAPACITY As Long = 4
Private Const MAX_ARRAY_LENGTH As Long = &H7FEFFFFF
Private Const OBJECT_REPR As String = "OBJECT"
Private Const MISSING_LONG As Long = -9999
Private Const CHR_QUOTE As String = """"
Private Const CHR_COMMA As String = ","
Public Enum ArrayTypes
    BA_UNDEFINED
    BA_UNALLOCATED
    BA_ONEDIMENSION
    BA_MULTIDIMENSION
    BA_JAGGED
End Enum
Public Enum ErrorCodes
    EC_START = vbObjectError + 512
    EC_EXPECTED_RANGE_OBJECT
    EC_EXPECTED_COLLECTION_OBJECT
    EC_MAX_DIMENSIONS_LIMIT
    EC_EXCEEDS_MAX_SORT_DEPTH
    EC_EXPECTED_JAGGED_ARRAY
    EC_EXPECTED_MULTIDIMENSION_ARRAY
    EC_EXPECTED_ARRAY
    EC_NULL_STRING
    EC_UNALLOCATED_ARRAY
    EC_UNDEFINED_ARRAY
    EC_INVALID_MULTIDIMENSIONAL_ARRAY_OPERATION
    EC_EXPECTED_VARIANT_ARRAY
    EC_EXCEEDS_MAX_ARRAY_LENGTH
    EC_STRING_TYPE_EXPECTED
    EC_CANNOT_CONVERT_TO_REQUESTED_STRUCTURE
    EC_CANNOT_SORT_OBJECTS
    EC_END
End Enum
Public Enum ComparisonType
    CT_EQUALITY
    CT_LIKENESS
End Enum
Public Enum SortMethods
    SM_TIMSORT
    SM_QUICKSORT_RECURSIVE
    SM_QUICKSORT_ITERATIVE
End Enum
Private Type ErrorDefinition
    Number          As Long
    Source          As String
    Description     As String
End Type
Private Type TFields
    Capacity        As Long
    Length          As Long
    LowerBound      As Long
    Items()         As Variant
    ArrayType       As ArrayTypes
    ErrorDefinitions(EC_START To EC_END) As ErrorDefinition
    LowerBoundSet   As Boolean
    SortMethod      As SortMethods
End Type
Private Type tString
    TEXT            As String
    Length          As Long
    ByteLength      As Long
End Type
Private This        As TFields
Private Sub Class_Initialize()
    Me.Capacity = DEFAULT_CAPACITY
    This.ArrayType = BA_UNALLOCATED
    PopulateErrorDefinitions
End Sub
Public Property Get Capacity() As Long
```

```
    Capacity = This.Capacity
End Property
Public Property Let Capacity(ByVal Value As Long)                                                             ...
    If Value < 0 Then Err.Raise 9
    If Value <> This.Capacity Then
        If This.Capacity > 0 Then
            If GetArrayLength(This.Items) <> Value Then
                Dim NewItems() As Variant
                NewItems = This.Items
                ReDim Preserve NewItems(This.LowerBound To (Value + This.LowerBound - 1))
                InternalItems = NewItems
            End If
        Else
            ReDim This.Items(This.LowerBound To (DEFAULT_CAPACITY + This.LowerBound - 1))
        End If
        This.Capacity = UBound(This.Items) - This.LowerBound + 1
    End If
End Property
Public Property Get Length() As Long                                                                          ...
    Length = This.Length
End Property
Public Property Get UpperBound() As Long                                                                      ...
    If This.ArrayType = BA_UNALLOCATED Then
        UpperBound = -1
    Else
        UpperBound = This.Length + This.LowerBound - 1
    End If
End Property
Public Property Get LowerBound() As Long                                                                      ...
    LowerBound = This.LowerBound
End Property
Public Property Let LowerBound(ByVal Value As Long)                                                           ...
    This.LowerBoundSet = True
    If Value <> This.LowerBound Then
        This.LowerBound = Value
        InternalItems = Rebase()
        This.Capacity = GetArrayLength(This.Items)
    End If
End Property
Public Property Get item(ByVal index As Long) As Variant                                                      ...
    If index <= This.Length Then
        If IsObject(This.Items(index)) Then
            Set item = This.Items(index)
        Else
            item = This.Items(index)
        End If
    Else
        Err.Raise 9
    End If
End Property
Public Property Let item(ByVal index As Long, ByVal Element As Variant)                                       ...
    If Me.UpperBound >= index Then
        If index < This.LowerBound Then
            Me.Unshift Element
        Else
            LetOrSetElement This.Items(index), Element
        End If
    Else
        Me.Push Element
    End If
End Property
Public Property Get Items() As Variant                                                                        ...
    Dim Result()    As Variant
    Result = InternalItems
    If This.ArrayType = ArrayTypes.BA_MULTIDIMENSION Then
        If IsJaggedArray(Result) Then Result = JaggedToMulti(Result)
    End If
    Items = Result
End Property
Public Property Let Items(ByVal Values As Variant)                                                            ...
    Const CONVERT_MD_TO_JAGGED As Boolean = True
    Const CONVERT_NESTED_JAGGED As Boolean = True
```

```
        Dim LocalLowerBound As Long
        Dim LocalValues() As Variant
        Dim TypeSet     As Boolean
        If TypeName(Values) = TypeName(Me) Then
            LocalValues = Values.Items
        ElseIf IsArray(Values) Then
            This.ArrayType = GetArrayType(Values)
            TypeSet = True
            If This.ArrayType = BA_UNALLOCATED Then
                LocalValues = GetEmptyArray
            Else
                LocalLowerBound = LBound(Values)
                If Not This.LowerBoundSet Then
                    This.LowerBound = LocalLowerBound
                End If
                LocalValues = ConvertArrayForStorage( _
                        Values, _
                        This.ArrayType, _
                        CONVERT_MD_TO_JAGGED, _
                        CONVERT_NESTED_JAGGED _
                        )
            End If
        Else
            If IsEmpty(Values) And This.ArrayType = ArrayTypes.BA_UNDEFINED Then
                RaiseError EC_EXPECTED_ARRAY, "Items", "Values"
            Else
                Me.clear.Push Values
            End If
            Exit Property
        End If
        If Not TypeSet Then
            This.ArrayType = GetArrayType(LocalValues)
        End If
        InternalItems = LocalValues
        This.Length = GetArrayLength(LocalValues)
        This.Capacity = This.Length
        If This.Capacity < DEFAULT_CAPACITY Then
            Me.Capacity = DEFAULT_CAPACITY
        End If
    End Property
    Public Property Get ArrayType() As ArrayTypes                                                   ...
        ArrayType = This.ArrayType
    End Property
    Public Property Let ArrayType(ByVal NewType As ArrayTypes)                                      ...
        Select Case NewType
            Case ArrayTypes.BA_UNDEFINED
                RaiseError EC_CANNOT_CONVERT_TO_REQUESTED_STRUCTURE, "ArrayType", "NewType"
            Case ArrayTypes.BA_UNALLOCATED
                If This.ArrayType <> BA_UNALLOCATED Then
                    Me.ResetToDefault
                End If
            Case ArrayTypes.BA_ONEDIMENSION
                Select Case This.ArrayType
                    Case ArrayTypes.BA_MULTIDIMENSION, ArrayTypes.BA_JAGGED
                        RaiseError EC_CANNOT_CONVERT_TO_REQUESTED_STRUCTURE, "ArrayType", "NewType"
                    Case Else
                        This.ArrayType = NewType
                End Select
            Case Else
                This.ArrayType = NewType
        End Select
    End Property
    Public Property Get SortMethod() As SortMethods                                                 ...
        SortMethod = This.SortMethod
    End Property
    Public Property Let SortMethod(ByVal Method As SortMethods)                                     ...
        This.SortMethod = Method
    End Property
    Private Property Get InternalItems() As Variant()                                               ...
        Dim Result()    As Variant
        If This.ArrayType <> BA_UNALLOCATED And This.ArrayType <> BA_UNDEFINED Then
            Result = This.Items
            If This.Capacity > This.Length Then
                If This.Length = 0 Then
                    ReDim Preserve Result(This.LowerBound To This.LowerBound)
                ElseIf This.Length > 0 Then
                    ReDim Preserve Result(This.LowerBound To Me.UpperBound)
                End If
            End If
        Else
            Result = GetEmptyArray
        End If
        InternalItems = Result
    End Property
    Private Property Let InternalItems(ByRef Value() As Variant)                                            ...
        This.Items = Value
    End Property
    Public Function Push(ParamArray Args() As Variant) As Variant                                           ...
        Dim Element     As Variant
        If This.ArrayType = ArrayTypes.BA_UNALLOCATED Or _
                This.ArrayType = ArrayTypes.BA_UNDEFINED Then
            This.ArrayType = ArrayTypes.BA_ONEDIMENSION
        End If
        For Each Element In Args
            If This.Length = This.Capacity Then
                EnsureCapacity This.Length + 1
            End If
            If IsArray(Element) Then
                Dim ArrayElement() As Variant
                Dim ArrayElementType As ArrayTypes
                ArrayElement = Element
                If This.ArrayType = BA_ONEDIMENSION Then
                    ArrayElementType = GetArrayType(ArrayElement)
                    If ArrayElementType = BA_MULTIDIMENSION Then
                        This.ArrayType = BA_MULTIDIMENSION
                    Else
                        This.ArrayType = BA_JAGGED
                    End If
                End If
                If LBound(ArrayElement) <> This.LowerBound Then
                    ArrayElement = Rebase(ArrayElement, ArrayElementType)
                End If
                LetOrSetElement This.Items(This.Length + This.LowerBound), ArrayElement
            Else
                LetOrSetElement This.Items(This.Length + This.LowerBound), Element
            End If
            inc This.Length
        Next
        Push = This.Length
    End Function
    Public Function Pop() As Variant                                                                        ...
        Dim Result      As Variant
        Dim NewItems()  As Variant
        If This.Length > 0 Then
            Result = This.Items(Me.UpperBound)
            NewItems = Me.Slice(This.LowerBound, Me.UpperBound)
            Me.Items = NewItems
        End If
        Pop = Result
    End Function
    Public Function Shift() As Variant                                                                      ...
        Dim NewItems()  As Variant
        Dim Result      As Variant
        If This.Length > 0 Then
            Result = This.Items(This.LowerBound)
            NewItems = Me.Slice(This.LowerBound + 1)
            Me.Items = NewItems
        End If
        Shift = Result
    End Function
    Public Function Unshift(ParamArray Args() As Variant) As Long                                           ...
        Dim NewItems()  As Variant
        Dim OldItems()  As Variant
        Dim OldType     As ArrayTypes
```

```
        NewItems = Args
        OldType = This.ArrayType
        OldItems = InternalItems
        Me.Items = NewItems
        If OldType <> BA_UNALLOCATED And OldType <> BA_UNDEFINED Then
            Me.Concat OldItems
            This.ArrayType = OldType
        End If
        Unshift = This.Length
    End Function
    Public Function ToString( _                                                                              ...
            Optional ByVal PrettyPrint As Boolean, _
            Optional ByVal Separator As String = CHR_COMMA, _
            Optional ByVal OpeningDelimiter As String = "{", _
            Optional ByVal ClosingDelimiter As String = "}", _
            Optional ByVal QuoteStrings As Boolean _
            ) As String
        Dim LocalArrayType As ArrayTypes
        Dim Result      As String
        Dim LocalItems() As Variant
        Dim Sep         As String
        Sep = IIf(PrettyPrint, Separator & Space(1), Separator)
        LocalItems = InternalItems
        LocalArrayType = GetArrayType(LocalItems)
        If Not LocalArrayType = ArrayTypes.BA_UNDEFINED And _
                Not LocalArrayType = ArrayTypes.BA_UNALLOCATED Then
            If LocalArrayType = ArrayTypes.BA_MULTIDIMENSION Then
                LocalItems = MultiToJagged(LocalItems)
            End If
            RecursiveToString _
                    SourceArray:=LocalItems, _
                    PrettyPrint:=PrettyPrint, _
                    Separator:=Sep, _
                    OpeningDelimiter:=OpeningDelimiter, _
                    ClosingDelimiter:=ClosingDelimiter, _
                    QuoteStrings:=QuoteStrings
            Result = StringBuilder(Final:=True)
        End If
        ToString = Result
    End Function
    Public Function Includes( _                                                                              ...
            ByVal SearchElement As Variant, _
            Optional ByVal FromIndex As Long = MISSING_LONG, _
            Optional ByVal Recurse As Boolean _
            ) As Boolean
        Dim LocalLength As Long
        Dim CurrentIndex As Long
        Dim SearchArray() As Variant
        SearchArray = InternalItems
        LocalLength = This.Length
        If LocalLength = 0 Then
            Includes = False
            Exit Function
        End If
        If FromIndex > This.LowerBound Then
            CurrentIndex = FromIndex
        ElseIf FromIndex = MISSING_LONG Then
            CurrentIndex = This.LowerBound
        Else
            CurrentIndex = LocalLength + FromIndex
        End If
        Includes = RecursiveIncludes(SearchElement, SearchArray, CurrentIndex, Recurse:=Recurse)
    End Function
    Public Function IncludesType( _                                                                          ...
            ByVal SearchTypeName As String, _
            Optional ByVal FromIndex As Long = MISSING_LONG, _
            Optional ByVal Recurse As Boolean _
            ) As Boolean
        Dim LocalLength As Long
        Dim CurrentIndex As Long
        Dim SearchArray() As Variant
        SearchArray = InternalItems
        LocalLength = This.Length
        If LocalLength = 0 Then
            IncludesType = False
            Exit Function
        End If
        If FromIndex > This.LowerBound Then
            CurrentIndex = FromIndex
        ElseIf FromIndex = MISSING_LONG Then
            CurrentIndex = This.LowerBound
        Else
            CurrentIndex = LocalLength + FromIndex
        End If
        IncludesType = RecursiveIncludes(SearchTypeName, SearchArray, CurrentIndex, True, Recurse)
    End Function
    Public Function Every( _                                                                        ...
            ByVal SearchElement As Variant, _
            Optional ByVal FromIndex As Long = MISSING_LONG _
            ) As Boolean
        Dim LocalLength As Long
        Dim CurrentIndex As Long
        Dim SearchArray() As Variant
        SearchArray = InternalItems
        LocalLength = This.Length
        If LocalLength = 0 Then
            Every = False
            Exit Function
        End If
        If FromIndex > This.LowerBound Then
            CurrentIndex = FromIndex
        ElseIf FromIndex = MISSING_LONG Then
            CurrentIndex = This.LowerBound
        Else
            CurrentIndex = LocalLength + FromIndex
        End If
        Every = RecursiveEvery(SearchElement, SearchArray, CurrentIndex)
    End Function
    Public Function EveryType( _                                                                    ...
            ByVal SearchTypeName As String, _
            Optional ByVal FromIndex As Long = MISSING_LONG _
            ) As Boolean
        Dim LocalLength As Long
        Dim CurrentIndex As Long
        Dim SearchArray() As Variant
        SearchArray = InternalItems
        LocalLength = This.Length
        If LocalLength = 0 Then
            EveryType = False
            Exit Function
        End If
        If FromIndex > This.LowerBound Then
            CurrentIndex = FromIndex
        ElseIf FromIndex = MISSING_LONG Then
            CurrentIndex = This.LowerBound
        Else
            CurrentIndex = LocalLength + FromIndex
        End If
        EveryType = RecursiveEvery(SearchTypeName, SearchArray, CurrentIndex, True)
    End Function
    Public Function Keys() As Variant()                                                             ...
        Dim i             As Long
        Dim LocalLowerBound As Long
        Dim Result()      As Variant
        If This.ArrayType = BA_UNDEFINED Then
            RaiseError EC_UNDEFINED_ARRAY, "Keys"
        ElseIf This.ArrayType = BA_UNALLOCATED Then
            RaiseError EC_UNALLOCATED_ARRAY, "Keys"
        Else
            LocalLowerBound = This.LowerBound
            ReDim Result(0 To This.Length - 1)
            For i = LBound(Result) To UBound(Result)
                Result(i) = i + LocalLowerBound
            Next
        End If
        Keys = Result
    End Function
    Public Function Max(ParamArray Args() As Variant) As Variant                                    ...
        Dim LocalItems() As Variant
        If UBound(Args) < LBound(Args) Then
            LocalItems = InternalItems
        Else
            LocalItems = Args
        End If
        Max = RecursiveMax(LocalItems)
    End Function
    Public Function Min(ParamArray Args() As Variant) As Variant                                    ...
        Dim LocalItems() As Variant
        If UBound(Args) < LBound(Args) Then
            LocalItems = InternalItems
        Else
            LocalItems = Args
        End If
        Min = RecursiveMin(LocalItems)
    End Function
    Public Function Slice( _                                                                        ...
            ByVal StartIndex As Long, _
            Optional ByVal EndIndex As Long = MISSING_LONG _
            ) As Variant()
        Dim LocalLength As Long
        Dim RelativeStart As Long
        Dim RelativeEnd As Long
        Dim OldIndex    As Long
        Dim Final       As Long
        Dim Count       As Long
        Dim NewIndex    As Long
        Dim LocalItems() As Variant
        Dim Result()    As Variant
        LocalItems = InternalItems
        LocalLength = This.Length
        RelativeStart = StartIndex
        If RelativeStart < LBound(LocalItems) Then
            If RelativeStart < 0 Then
                OldIndex = Max((LocalLength + RelativeStart), LBound(LocalItems))
            Else
                OldIndex = Max((LocalLength - RelativeStart), LBound(LocalItems))
            End If
        Else
            OldIndex = Min(RelativeStart, LocalLength)
        End If
        If EndIndex = MISSING_LONG Then
            RelativeEnd = LocalLength + LBound(LocalItems)
        Else
            RelativeEnd = EndIndex
        End If
        If RelativeEnd < LBound(LocalItems) Then
            Final = Max((LocalLength + RelativeEnd), LBound(LocalItems))
        Else
            Final = Min(RelativeEnd, LocalLength + LBound(LocalItems))
        End If
        NewIndex = LBound(LocalItems)
        Count = Max(Final - OldIndex, 0) + LBound(LocalItems)
        If Count > NewIndex Then
            ReDim Result(NewIndex To Count - 1)
            Do While OldIndex < Final
                If OldIndex >= LBound(LocalItems) And OldIndex <= UBound(LocalItems) Then
                    LetOrSetElement Result(NewIndex), LocalItems(OldIndex)
                    inc NewIndex
                    inc OldIndex
                End If
            Loop
            If This.ArrayType = BA_MULTIDIMENSION Then
                Slice = JaggedToMulti(Result)
            Else
                Slice = Result
            End If
        End If
    End Function
    Public Function FromExcelRange( _                                                                       ...
            ByRef FromRange As Object, _
            Optional ByVal DetectLastRow As Boolean, _
            Optional ByVal DetectLastColumn As Boolean _
            ) As BetterArray
        If TypeName(FromRange) = "Range" Then
            Dim StartColumn As Long
            Dim EndColumn As Long
            Dim StartRow As Long
            Dim EndRow  As Long
            With FromRange
                StartColumn = .Column
                StartRow = .Row
                EndColumn = .Column + .Columns.Count - 1
                EndRow = .Row + .rows.Count - 1
            End With
            With FromRange.Parent
                If DetectLastColumn Then
                    EndColumn = .Cells.item(StartRow, .Columns.Count).End(xlToLeft).Column
                End If
                If DetectLastRow Then
                    EndRow = .Cells.item(.rows.Count, StartColumn).End(xlUp).Row
                End If
                Me.Items = .Range(.Cells(StartRow, StartColumn), .Cells(EndRow, EndColumn)).Value
            End With
            If StartColumn = EndColumn And StartRow <> EndRow Then
                Me.Items = Me.ExtractSegment(, StartColumn)
            ElseIf StartColumn <> EndColumn And StartRow = EndRow Then
                Me.Items = Me.ExtractSegment(StartRow)
            End If
        Else
            RaiseError ErrorCodes.EC_EXPECTED_RANGE_OBJECT, "FromExcelRange()", "FromRange"
        End If
        Set FromExcelRange = Me
    End Function
    Public Function ExtractSegment( _                                                                       ...
            Optional ByVal RowIndex As Long = MISSING_LONG, _
            Optional ByVal ColumnIndex As Long = MISSING_LONG _
            ) As Variant()
        Dim i           As Long
        Dim LocalRowIndex As Long
        Dim LocalColumnIndex As Long
        Dim NestedBounds() As Long
        Dim LocalItems() As Variant
        Dim Result()    As Variant
        LocalItems = InternalItems
        If RowIndex = MISSING_LONG Then
            If ColumnIndex = MISSING_LONG Then
                Result = Me.Items
            Else
                Select Case This.ArrayType
                    Case BA_ONEDIMENSION
                        If ColumnIndex >= LBound(LocalItems) And ColumnIndex <= UBound(LocalItems) Then
                            LocalColumnIndex = ColumnIndex
                        Else
                            LocalColumnIndex = LBound(LocalItems)
                        End If
                        Result = Array(LocalItems(LocalColumnIndex))
                    Case BA_JAGGED, BA_MULTIDIMENSION
                        NestedBounds = GetMaxBoundsAtDimension(LocalItems, 2)
                        If ColumnIndex >= NestedBounds(0) And ColumnIndex <= NestedBounds(1) Then
                            LocalColumnIndex = ColumnIndex
                        Else
                            LocalColumnIndex = This.LowerBound
                        End If
                        ReDim Result(LBound(LocalItems) To UBound(LocalItems))
                        For i = LBound(LocalItems) To UBound(LocalItems)
                            Result(i) = LocalItems(i)(LocalColumnIndex)
                        Next
                    Case BA_UNALLOCATED
```

```
                    Result = LocalItems
                Case Else
            End Select
        End If
    Else
        If RowIndex >= LBound(LocalItems) And RowIndex <= UBound(LocalItems) Then
            LocalRowIndex = RowIndex
        Else
            LocalRowIndex = LBound(LocalItems)
        End If
        If ColumnIndex = MISSING_LONG Then
            Select Case This.ArrayType
                Case BA_ONEDIMENSION
                    Result = Array(LocalItems(LocalRowIndex))
                Case BA_JAGGED, BA_MULTIDIMENSION
                    Result = LocalItems(LocalRowIndex)
                Case BA_UNALLOCATED
                    Result = LocalItems
                Case Else
            End Select
        Else
            Select Case This.ArrayType
                Case BA_ONEDIMENSION
                    Result = Array(LocalItems(LocalRowIndex))
                Case BA_JAGGED, BA_MULTIDIMENSION
                    NestedBounds = GetMaxBoundsAtDimension(LocalItems, 2)
                    If ColumnIndex >= NestedBounds(0) And ColumnIndex <= NestedBounds(1) Then
                        LocalColumnIndex = ColumnIndex
                    Else
                        LocalColumnIndex = This.LowerBound
                    End If
                    If IsArray(LocalItems(LocalRowIndex)(LocalColumnIndex)) Then
                        Result = LocalItems(LocalRowIndex)(LocalColumnIndex)
                    Else
                        Result = Array(LocalItems(LocalRowIndex)(LocalColumnIndex))
                    End If
                Case BA_UNALLOCATED
                    Result = LocalItems
                Case Else
            End Select
        End If
    End If
    ExtractSegment = Result
End Function
Public Function ToExcelRange( _                                                                    ...
        ByRef Destination As Object, _
        Optional ByVal TransposeValues As Boolean _
        ) As Object
    Const TARGET_APPLICATION As String = "Microsoft Excel"
    Const TARGET_OBJECT As String = "Range"
    Dim LocalRange  As Object
    Dim LocalItems() As Variant
    Dim Depth       As Long
    Dim LengthRows  As Long
    Dim LengthColumns As Long
    Dim AvailableRows As Long
    Dim AvailableColumns As Long
    Dim DestType    As String
    Dim DestApplication As String
    On Error Resume Next
    DestType = TypeName(Destination)
    DestApplication = Destination.Application.Name
    On Error GoTo 0
    If DestType = TARGET_OBJECT And DestApplication = TARGET_APPLICATION Then
        AvailableRows = Destination.Parent.rows.Count - Destination.Row + 1
        AvailableColumns = Destination.Parent.Columns.Count - Destination.Column + 1
        LocalItems = InternalItems
        Depth = GetJaggedArrayDepth(LocalItems)
        If Depth > 0
            If Depth = 1 Then
                LocalItems = ConvertOneDimensionArrayToJagged(LocalItems)
            End If
```

```
                Const OutputDepth As
                LocalItems = JaggedToMulti(LocalItems, OutputDepth, EnsureScalar:=True)
                If TransposeValues Then
                    LocalItems = Transpose2DArray(LocalItems)
                End If
                LocalItems = TrimColumnsMultidimensionArray(LocalItems, AvailableColumns)
                LocalItems = TrimRowsMultidimensionArray(LocalItems, AvailableRows)
                LengthRows = UBound(LocalItems, 1) - LBound(LocalItems, 1) + 1
                LengthColumns = UBound(LocalItems, 2) - LBound(LocalItems, 2) + 1
                Set LocalRange = Destination.RESIZE( _
                        rowSize:=LengthRows, _
                        ColumnSize:=LengthColumns _
                        )
                LocalRange.Value = LocalItems
            Else
            End If
        Else
            RaiseError ErrorCodes.EC_EXPECTED_RANGE_OBJECT, "ToExcelRange()", "Destination"
        End If
        Set ToExcelRange = LocalRange
    End Function
    Public Function IsSorted(Optional ByVal ColumnIndex As Long = MISSING_LONG) As Boolean                                    ...
        Dim i           As Long
        Dim LocalLowerBound As Long
        Dim LocalUpperBound As Long
        Dim LocalColumnIndex As Long
        Dim Depth       As Long
        Dim Result      As Boolean
        Dim LocalItems() As Variant
        Dim StoredType  As ArrayTypes
        Result = True
        LocalItems = InternalItems
        LocalLowerBound = LBound(LocalItems)
        LocalUpperBound = UBound(LocalItems)
        StoredType = GetArrayType(LocalItems)
        If StoredType <> BA_UNDEFINED And StoredType <> BA_UNALLOCATED Then
            Select Case StoredType
                Case BA_ONEDIMENSION
                    For i = LocalLowerBound To LocalUpperBound - 1
                        If LocalItems(i) > LocalItems(i + 1) Then
                            Result = False
                            Exit For
                        End If
                    Next
                Case BA_MULTIDIMENSION
                    If ColumnIndex = MISSING_LONG Then
                        LocalColumnIndex = LBound(LocalItems, 2)
                    Else
                        LocalColumnIndex = CLng(ColumnIndex)
                    End If
                    For i = LocalLowerBound To LocalUpperBound - 1
                        If LocalItems(i, LocalColumnIndex) > LocalItems(i + 1, LocalColumnIndex) Then
                            Result = False
                            Exit For
                        End If
                    Next
                Case BA_JAGGED
                    Depth = GetJaggedArrayDepth(LocalItems)
                    If Depth > 2 Then
                        IsSorted = False
                        RaiseError EC_EXCEEDS_MAX_SORT_DEPTH, "IsSorted"
                    Else
                        For i = LocalLowerBound To LocalUpperBound - 1
                            If i = LocalLowerBound Then
                                If ColumnIndex = MISSING_LONG Then
                                    LocalColumnIndex = LBound(LocalItems(i))
                                Else
                                    LocalColumnIndex = CLng(ColumnIndex)
                                End If
                            End If
                            If LocalItems(i)(LocalColumnIndex) > LocalItems(i + 1)(LocalColumnIndex)  _
                            Then
                                Result = False
                                Exit For
                            End If
                        Next
                    End If
            End Select
        End If
        IsSorted = Result
    End Function
    Public Function IndexOf( _                                                                 ...
            ByVal SearchElement As Variant, _
            Optional ByVal FromIndex As Long = MISSING_LONG, _
            Optional ByVal CompType As ComparisonType _
            ) As Long
        Dim LocalItems() As Variant
        Dim RelativeStart As Long
        Dim CurrentIndex As Long
        If CompType = CT_LIKENESS Then
            If TypeName(SearchElement) <> "String" Then
                RaiseError EC_STRING_TYPE_EXPECTED, "IndexOf", "searchElement"
            End If
        End If
        If This.Length = 0 Then
            IndexOf = MISSING_LONG
            Exit Function
        End If
        LocalItems = InternalItems
        If FromIndex = MISSING_LONG Then
            RelativeStart = LBound(LocalItems)
        Else
            RelativeStart = FromIndex
        End If
        If RelativeStart >= LBound(LocalItems) Then
            CurrentIndex = RelativeStart
        Else
            If RelativeStart > 0 Then
                CurrentIndex = LBound(LocalItems)
            Else
                CurrentIndex = UBound(LocalItems) + RelativeStart
            End If
            If CurrentIndex < LBound(LocalItems) Then
                CurrentIndex = LBound(LocalItems)
            End If
        End If
        Dim IsMatch     As Boolean
        Do While CurrentIndex <= UBound(LocalItems)
            Select Case CompType
                Case ComparisonType.CT_LIKENESS
                    IsMatch = CStr(LocalItems(CurrentIndex)) Like CStr(SearchElement)
                Case Else
                    IsMatch = ElementsAreEqual(SearchElement, LocalItems(CurrentIndex))
            End Select
            If IsMatch Then
                IndexOf = CurrentIndex
                Exit Function
            End If
            inc CurrentIndex
        Loop
        IndexOf = MISSING_LONG
    End Function
    Public Function LastIndexOf( _                                                             ...
            ByVal SearchElement As Variant, _
            Optional ByVal FromIndex As Long = MISSING_LONG, _
            Optional ByVal CompType As ComparisonType _
            ) As Long
        Dim LocalItems() As Variant
        Dim CurrentIndex As Long
        If CompType = CT_LIKENESS Then
            If TypeName(SearchElement) <> "String" Then
                RaiseError EC_STRING_TYPE_EXPECTED, "IndexOf", "searchElement"
            End If
        End If
```

```
        If This.Length = 0 Then
            LastIndexOf = MISSING_LONG
            Exit Function
        End If
        LocalItems = InternalItems
        If FromIndex = MISSING_LONG Then
            CurrentIndex = UBound(LocalItems)
        Else
            If FromIndex >= LBound(LocalItems) Then
                CurrentIndex = Min(FromIndex, UBound(LocalItems))
            ElseIf FromIndex < 0 Then
                CurrentIndex = UBound(LocalItems) + FromIndex
            End If
        End If
        Dim IsMatch     As Boolean
        Do While CurrentIndex >= LBound(LocalItems)
            Select Case CompType
                Case ComparisonType.CT_LIKENESS
                    IsMatch = CStr(LocalItems(CurrentIndex)) Like CStr(SearchElement)
                Case Else
                    IsMatch = ElementsAreEqual(SearchElement, LocalItems(CurrentIndex))
            End Select
            If IsMatch Then
                LastIndexOf = CurrentIndex
                Exit Function
            End If
            dec CurrentIndex
        Loop
        LastIndexOf = MISSING_LONG
    End Function
    Public Function Remove(ByVal index As Long) As Long                                    ...
        Dim RelativeIndex As Long
        Dim LocalType   As ArrayTypes
        RelativeIndex = MISSING_LONG
        LocalType = This.ArrayType
        If index >= This.LowerBound Then
            If index <= Me.UpperBound Then
                RelativeIndex = index
            End If
        Else
            If index < 0 Then
                RelativeIndex = Me.UpperBound + index
                If RelativeIndex < This.LowerBound Then RelativeIndex = MISSING_LONG
            End If
        End If
        If RelativeIndex <> MISSING_LONG Then
            Dim BeforeSlice() As Variant
            Dim AfterSlice() As Variant
            Dim BeforeExists As Boolean
            Dim AfterExists As Boolean
            If RelativeIndex > This.LowerBound Then
                BeforeSlice = Me.Slice(This.LowerBound, RelativeIndex)
                BeforeExists = True
            End If
            If RelativeIndex < Me.UpperBound Then
                AfterSlice = Me.Slice(RelativeIndex + 1)
                AfterExists = True
            End If
            If BeforeExists Then
                If AfterExists Then
                    Me.Items = InternalConcat(BeforeSlice, AfterSlice)
                Else
                    Me.Items = BeforeSlice
                End If
            Else
                If AfterExists Then
                    Me.Items = AfterSlice
                End If
            End If
            If BeforeExists Or AfterExists Then
                If This.ArrayType = BA_JAGGED And LocalType = BA_MULTIDIMENSION Then
                    This.ArrayType = LocalType
                End If
            Else
                Me.clear
            End If
        End If
        Remove = This.Length
    End Function
    Public Function Splice( _                                                                        ...
            ByVal StartIndex As Long, _
            ParamArray Args() As Variant _
            ) As Variant()
        Dim LocalItems() As Variant
        Dim ActualStart As Long
        Dim ActualDeleteCount As Long
        Dim ArgsCount   As Long
        Dim LocalLength As Long
        Dim LocalArgs() As Variant
        Dim Result()    As Variant
        Dim i           As Long
        Dim TempArray() As Variant
        Dim ItemCount   As Long
        Dim TempItems() As Variant
        LocalArgs = Args
        LocalItems = InternalItems
        LocalLength = UBound(LocalItems) + 1
        If StartIndex < LBound(LocalItems) Then
            ActualStart = Max(LocalLength + StartIndex, LBound(LocalItems))
        Else
            ActualStart = Min(StartIndex, LocalLength)
        End If
        ArgsCount = GetArrayLength(LocalArgs)
        If ArgsCount = 0 Then
            ItemCount = 0
            ActualDeleteCount = LocalLength - ActualStart
        Else
            ItemCount = ArgsCount - 1
            ActualDeleteCount = Min(Max(Args(LBound(Args)), 0), LocalLength - ActualStart)
        End If
        If LocalLength + ItemCount - ActualDeleteCount > MAX_ARRAY_LENGTH Then
            RaiseError EC_EXCEEDS_MAX_ARRAY_LENGTH, "Splice"
            Exit Function
        End If
        If ActualDeleteCount > 0 Then
            ReDim Result(0 To ActualDeleteCount - 1)
            i = 0
            Do While i < ActualDeleteCount
                LetOrSetElement Result(i), LocalItems(ActualStart + i)
                inc i
            Loop
        Else
            ReDim Result(0)
        End If
        If ItemCount > 0 Then
            ReDim TempItems(0 To ItemCount - 1)
            For i = LBound(TempItems) To UBound(TempItems)
                TempItems(i) = Args(i + 1)
            Next
        End If
        If ItemCount < ActualDeleteCount Then
            i = ActualStart
            Do While i < (LocalLength - ActualDeleteCount)
                LetOrSetElement LocalItems(i + ItemCount), LocalItems(i + ActualDeleteCount)
                inc i
            Loop
            i = LocalLength
            Dim TempBetterArray As BetterArray
            Set TempBetterArray = New BetterArray
            TempBetterArray.Items = LocalItems
            Do While i > (LocalLength - ActualDeleteCount + ItemCount)
                TempBetterArray.Remove (i - 1)
                dec i
            Loop
```

```
            LocalItems = TempBetterArray.Items
            Set TempBetterArray = Nothing
        ElseIf ItemCount > ActualDeleteCount Then
            i = (LocalLength - ActualDeleteCount)
            TempArray = LocalItems
            ReDim Preserve TempArray(This.LowerBound To i + ItemCount - 1) As Variant
            Do While i > ActualStart
                LetOrSetElement TempArray(i + ItemCount - 1), LocalItems(i + ActualDeleteCount - 1)
                dec i
            Loop
            LocalItems = TempArray
        End If
        If ItemCount > 0 Then
            i = ActualStart
            Dim j       As Long
            For j = LBound(TempItems) To UBound(TempItems)
                LocalItems(i) = TempItems(j)
                inc i
            Next
        End If
        InternalItems = LocalItems
        Splice = Result
    End Function
    Public Function Fill( _                                                                   ...
            ByVal Value As Variant, _
            Optional ByVal StartIndex As Long = MISSING_LONG, _
            Optional ByVal EndIndex As Long = MISSING_LONG _
            ) As BetterArray
        Dim LocalItems() As Variant
        Dim RelativeStart As Long
        Dim RelativeEnd As Long
        LocalItems = InternalItems
        If StartIndex = MISSING_LONG Then
            RelativeStart = LBound(LocalItems)
        Else
            If StartIndex < 0 Then
                RelativeStart = Max(Me.UpperBound + StartIndex, LBound(LocalItems))
            ElseIf StartIndex < LBound(LocalItems) Then
                RelativeStart = LBound(LocalItems)
            Else
                RelativeStart = Min(StartIndex, Me.UpperBound)
            End If
        End If
        If EndIndex = MISSING_LONG Then
            RelativeEnd = Me.UpperBound
        Else
            If EndIndex < 0 And EndIndex < LBound(LocalItems) Then
                RelativeEnd = Max(Me.UpperBound + EndIndex, LBound(LocalItems))
            ElseIf EndIndex < LBound(LocalItems) Then
                RelativeEnd = Me.UpperBound
            Else
                RelativeEnd = Min(EndIndex, Me.UpperBound)
            End If
        End If
        InternalItems = RecursiveFill(LocalItems, Value, RelativeStart, RelativeEnd)
        Set Fill = Me
    End Function
    Public Function ParseFromString( _                                                        ...
            ByVal SourceString As String, _
            Optional ByVal ValueSeparator As String = CHR_COMMA, _
            Optional ByVal ArrayOpenDelimiter As String, _
            Optional ByVal ArrayClosingDelimiter As String _
            ) As BetterArray
        Dim Opener      As String
        Dim Closer      As String
        Dim ArraysAreDelimited As Boolean
        Dim Result()    As Variant
        If Len(SourceString) > 0 Then
            If ArrayOpenDelimiter = vbNullString And ArrayClosingDelimiter = vbNullString Then
                Dim FirstChar As String
                Dim LastChar As String
                FirstChar = Left$(SourceString, 1)
                LastChar = Right$(SourceString, 1)
                If (Asc(LastChar) - Asc(FirstChar) < 3) And _
                        (Asc(FirstChar) < 65 Or Asc(FirstChar) > 90) And _
                        (Asc(FirstChar) < 97 Or Asc(FirstChar) > 122) And _
                        Not IsNumeric(FirstChar) Then
                    Opener = FirstChar
                    Closer = LastChar
                    ArraysAreDelimited = True
                End If
            Else
                Opener = ArrayOpenDelimiter
                Closer = ArrayClosingDelimiter
                ArraysAreDelimited = True
            End If
            If ArraysAreDelimited Then
                Result = ParseDelimitedArrayString(SourceString, ValueSeparator, Opener, Closer)
            Else
                Result = ParseArraySegmentFromString(SourceString, 1, 0, ValueSeparator)
            End If
        Else
            RaiseError EC_NULL_STRING, "ParseFromString", "SourceString"
        End If
        Me.Items = Result
        Set ParseFromString = Me
    End Function
    Public Function Transpose() As BetterArray                                                   ...
        Dim Result()    As Variant
        Dim LocalItems() As Variant
        Dim LocalType   As ArrayTypes
        Dim StoredType  As ArrayTypes
        StoredType = This.ArrayType
        LocalItems = InternalItems
        LocalType = GetArrayType(LocalItems)
        Select Case LocalType
            Case ArrayTypes.BA_ONEDIMENSION
                Result = Transpose1DArray(LocalItems)
            Case ArrayTypes.BA_MULTIDIMENSION
                Result = Transpose2DArray(LocalItems)
            Case ArrayTypes.BA_JAGGED
                Result = TransposeArrayOfArrays(LocalItems)
            Case Else
                Result = LocalItems
        End Select
        Me.Items = Result
        If StoredType = BA_MULTIDIMENSION Then
            LocalType = GetArrayType(Result)
            If LocalType = BA_JAGGED Then
                This.ArrayType = BA_MULTIDIMENSION
            End If
        End If
        Set Transpose = Me
    End Function
    Public Function Clone() As BetterArray                                                       ...
        Dim Result      As BetterArray
        Set Result = New BetterArray
        Result.LowerBound = This.LowerBound
        Result.Items = Me.Items
        Set Clone = Result
    End Function
    Public Function ResetToDefault() As BetterArray                                              ...
        This.LowerBound = 0
        ReDim This.Items(This.LowerBound To DEFAULT_CAPACITY + This.LowerBound)
        This.Length = 0
        Me.Capacity = DEFAULT_CAPACITY
        This.ArrayType = BA_UNALLOCATED
        Set ResetToDefault = Me
    End Function
    Public Function clear() As BetterArray                                                       ...
        Dim OldCapacity As Long
        OldCapacity = This.Capacity
        ReDim This.Items(This.LowerBound To OldCapacity + This.LowerBound)
        This.ArrayType = BA_UNALLOCATED
        This.Length = 0
        Me.Capacity = OldCapacity
        Set clear = Me
    End Function
    Public Function Concat(ParamArray Args() As Variant) As BetterArray                                    ...
        Dim i           As Long
        Dim Result()    As Variant
        Dim Stored()    As Variant
        Dim StoredType  As ArrayTypes
        Dim StoredCapacity As Long
        StoredType = This.ArrayType
        StoredCapacity = This.Capacity
        If StoredType <> BA_JAGGED And StoredType <> BA_MULTIDIMENSION Then
            For i = LBound(Args) To UBound(Args)
                If IsArray(Args(i)) Then
                    If IsMultidimensionalArray(Args(i)) Then
                        StoredType = BA_MULTIDIMENSION
                        Exit For
                    End If
                End If
            Next
        End If
        If This.ArrayType <> BA_UNALLOCATED Then
            Stored = InternalItems
        End If
        Result = ConcatDelegate(Stored, Args)
        Me.Items = Result
        If StoredType = BA_MULTIDIMENSION Then
            This.ArrayType = StoredType
        End If
        If This.Capacity < StoredCapacity Then
            Me.Capacity = StoredCapacity
        End If
        Set Concat = Me
    End Function
    Public Function CopyFromCollection(ByVal SourceCollection As Collection) As BetterArray                 ...
        If SourceCollection Is Nothing Then
            RaiseError EC_EXPECTED_COLLECTION_OBJECT, "CopyFromCollection", "SourceCollection"
        End If
        Dim i           As Long
        Dim NewItems()  As Variant
        This.Length = SourceCollection.Count
        If This.Length = 0 Then
            NewItems = GetEmptyArray
        Else
            ReDim NewItems(This.LowerBound To (This.Length - This.LowerBound - 1))
            For i = 1 To This.Length
                NewItems(i + This.LowerBound - 1) = SourceCollection.item(i)
            Next
        End If
        Me.Items = NewItems
        Set CopyFromCollection = Me
    End Function
    Public Function Sort(Optional ByVal SortColumn As Long = MISSING_LONG) As BetterArray                   ...
        Dim LocalItems() As Variant
        Dim SortedItems() As Variant
        Dim LocalArrayType As ArrayTypes
        Dim LocalSortColumn As Long
        Dim FirstChildLowerBound As Long
        Set Sort = Me
        If Me.IncludesType("Object") Then
            RaiseError EC_CANNOT_SORT_OBJECTS, "Sort"
            Exit Function
        End If
        LocalItems = InternalItems
        LocalArrayType = GetArrayType(LocalItems)
        If LocalArrayType = ArrayTypes.BA_UNALLOCATED Or _
                LocalArrayType = ArrayTypes.BA_UNDEFINED Then
            SortedItems = GetEmptyArray
        Else
            If This.Length > 0 Then
                If LocalArrayType <> BA_ONEDIMENSION Then
                    If LocalArrayType = ArrayTypes.BA_MULTIDIMENSION Then
                        LocalItems = MultiToJagged(LocalItems)
                    End If
                    Dim Depth As Long
                    Depth = GetJaggedArrayDepth(LocalItems)
                    If Depth > 2 Then
                        RaiseError EC_EXCEEDS_MAX_SORT_DEPTH, "Sort"
                    End If
                    FirstChildLowerBound = LBound(LocalItems(LBound(LocalItems)))
                    If SortColumn = MISSING_LONG Then
                        LocalSortColumn = FirstChildLowerBound
                    Else
                        LocalSortColumn = SortColumn - 1 + FirstChildLowerBound
                    End If
                End If
                ApplySortMethod _
                        LocalItems, _
                        LocalArrayType, _
                        LocalSortColumn
            End If
            SortedItems = LocalItems
        End If
        Me.Items = SortedItems
    End Function
    Public Function CopyWithin( _                                                              ...
            ByVal Target As Long, _
            Optional ByVal StartIndex As Long = MISSING_LONG, _
            Optional ByVal EndIndex As Long = MISSING_LONG _
            ) As BetterArray
        Dim LocalLength As Long
        Dim RelativeTarget As Long
        Dim RelativeStart As Long
        Dim RelativeEnd As Long
        Dim ToIndex     As Long
        Dim FromIndex   As Long
        Dim Final       As Long
        Dim Count       As Long
        Dim Direction   As Long
        Dim LocalItems() As Variant
        Select Case This.ArrayType
            Case ArrayTypes.BA_UNDEFINED
                RaiseError EC_EXPECTED_ARRAY, "CopyWithin"
            Case ArrayTypes.BA_UNALLOCATED
                RaiseError EC_UNALLOCATED_ARRAY, "CopyWithin"
            Case Else
                LocalItems = InternalItems
                LocalLength = This.Length
                RelativeTarget = Target
                If RelativeTarget < 0 Then
                    ToIndex = Max((LocalLength + RelativeTarget), 0)
                Else
                    ToIndex = Min(RelativeTarget, LocalLength)
                End If
                RelativeStart = IIf(StartIndex = MISSING_LONG, LBound(LocalItems), StartIndex)
                If RelativeStart < 0 Then
                    FromIndex = Max((LocalLength + RelativeStart), 0)
                Else
                    FromIndex = Min(RelativeStart, LocalLength)
                End If
                If EndIndex = MISSING_LONG Then
                    RelativeEnd = LocalLength
                Else
                    RelativeEnd = EndIndex
                End If
                If RelativeEnd < 0 Then
                    Final = Max((LocalLength + RelativeEnd), 0)
                Else
                    Final = Min(RelativeEnd, LocalLength)
                End If
                Count = Min(Final - FromIndex, LocalLength - ToIndex)
                If FromIndex < ToIndex And ToIndex < FromIndex + Count Then
                    Direction = -1
                    inc FromIndex, Count - 1
                    inc ToIndex, Count - 1
                Else
                    Direction = 1
                End If
                Do While Count > 0
                    If FromIndex >= LBound(LocalItems) And FromIndex <= UBound(LocalItems) Then
                        LocalItems(ToIndex) = LocalItems(FromIndex)
                    End If
                    inc FromIndex, Direction
                    inc ToIndex, Direction
                    dec Count
                Loop
                Me.Items = LocalItems
        End Select
        Set CopyWithin = Me
    End Function
    Public Function Filter( _                                                                                          ...
            ByVal Match As Variant, _
            Optional ByVal Include As Boolean, _
            Optional ByVal Recurse As Boolean) As BetterArray
        Dim Result()    As Variant
        Dim LocalItems() As Variant
        Dim LocalType   As ArrayTypes
        LocalType = This.ArrayType
        If LocalType = BA_UNDEFINED Then
            RaiseError EC_EXPECTED_ARRAY, "Filter"
        ElseIf LocalType = BA_UNALLOCATED Then
            Result = GetEmptyArray
        Else
            LocalItems = InternalItems
            Result = RecursiveFilter( _
                    LocalItems, _
                    Match, _
                    Include, _
                    Recurse _
                    )
        End If
        Me.Items = Result
        If LocalType = BA_MULTIDIMENSION Then This.ArrayType = LocalType
        Set Filter = Me
    End Function
    Public Function FilterType( _                                                                                      ...
            ByVal SearchTypeName As String, _
            Optional ByVal Include As Boolean, _
            Optional ByVal Recurse As Boolean) As BetterArray
        Dim Result()    As Variant
        Dim LocalItems() As Variant
        Dim LocalType   As ArrayTypes
        LocalType = This.ArrayType
        If LocalType = BA_UNDEFINED Then
            RaiseError EC_EXPECTED_ARRAY, "Filter"
        ElseIf LocalType = BA_UNALLOCATED Then
            Result = GetEmptyArray
        Else
            LocalItems = InternalItems
            Result = RecursiveFilter( _
                    LocalItems, _
                    SearchTypeName, _
                    Include, _
                    Recurse, _
                    True _
                    )
        End If
        Me.Items = Result
        If LocalType = BA_MULTIDIMENSION Then This.ArrayType = LocalType
        Set FilterType = Me
    End Function
    Public Function Reverse(Optional ByVal Recurse As Boolean) As BetterArray                                          ...
        Dim LocalItems() As Variant
        Dim Result()    As Variant
        Dim CurrentType As ArrayTypes
```

```
        CurrentType = This.ArrayType
        Select Case CurrentType
            Case BA_UNDEFINED
                RaiseError EC_UNDEFINED_ARRAY, "Reverse"
            Case BA_ONEDIMENSION, BA_MULTIDIMENSION, BA_JAGGED
                LocalItems = InternalItems
                Result = RecursiveReverse(LocalItems, Recurse)
                Me.Items = Result
                This.ArrayType = CurrentType
        End Select
        Set Reverse = Me
    End Function
    Public Function Flatten() As BetterArray                                                          ...
        Dim LocalItems() As Variant
        Dim Result      As BetterArray
        Select Case This.ArrayType
            Case BA_UNDEFINED
                RaiseError EC_UNDEFINED_ARRAY, "Flatten"
            Case BA_MULTIDIMENSION, BA_JAGGED
                LocalItems = InternalItems
                Set Result = New BetterArray
                Result.LowerBound = This.LowerBound
                RecursiveFlatten LocalItems, Result
                Me.Items = Result.Items
        End Select
        Set Flatten = Me
    End Function
    Public Function Shuffle(Optional ByVal Recurse As Boolean) As BetterArray                         ...
        Dim LocalItems() As Variant
        Dim Result()    As Variant
        Dim CurrentType As ArrayTypes
        CurrentType = This.ArrayType
        Select Case CurrentType
            Case BA_UNDEFINED
                RaiseError EC_UNDEFINED_ARRAY, "Shuffle"
            Case BA_ONEDIMENSION, BA_MULTIDIMENSION, BA_JAGGED
                LocalItems = InternalItems
                Result = RecursiveShuffle(LocalItems, Recurse)
                Me.Items = Result
                This.ArrayType = CurrentType
        End Select
        Set Shuffle = Me
    End Function
    Public Function Unique(Optional ByVal ColumnIndex As Long = MISSING_LONG) As BetterArray         ...
        Dim LocalItems() As Variant
        Dim LocalType   As ArrayTypes
        Dim Result      As BetterArray
        Dim i           As Long
        LocalItems = InternalItems
        LocalType = This.ArrayType
        Set Result = New BetterArray
        Result.LowerBound = Me.LowerBound
        If (LocalType = BA_JAGGED Or LocalType = BA_MULTIDIMENSION) And ColumnIndex <> MISSING_LONG Then
            Dim LocalDepth As Long
            LocalDepth = GetJaggedArrayDepth(LocalItems)
            If LocalDepth = 2 Then
                Dim LocalColumn As Long
                Dim ComparisonValues() As Variant
                Dim Comparator As BetterArray
                Dim Bounds() As Long
                Set Comparator = New BetterArray
                Bounds = GetMaxBoundsAtDimension(LocalItems, 2)
                Comparator.LowerBound = Bounds(0)
                LocalColumn = ColumnIndex - 1 + Bounds(0)
                If LocalColumn < Bounds(0) Or LocalColumn > Bounds(1) Then
                    LocalColumn = Bounds(0)
                End If
                ComparisonValues = Me.ExtractSegment(ColumnIndex:=LocalColumn)
                Comparator.Items = ComparisonValues
                For i = LBound(LocalItems) To UBound(LocalItems) Then
                    If Comparator.IndexOf(LocalItems(i)(LocalColumn)) = i Then
                        Result.Push LocalItems(i)
                    End If
                Next
            Else
                For i = LBound(LocalItems) To UBound(LocalItems)
                    If Me.IndexOf(LocalItems(i)) = i Then
                        Result.Push LocalItems(i)
                    End If
                Next
            End If
        Else
            For i = LBound(LocalItems) To UBound(LocalItems)
                If Me.IndexOf(LocalItems(i)) = i Then
                    Result.Push LocalItems(i)
                End If
            Next
        End If
        Me.Items = Result.Items
        This.ArrayType = LocalType
        Set Unique = Me
    End Function
    Public Function FromCSVFile( _                                                              ...
            ByVal Path As String, _
            Optional ByVal ColumnDelimiter As String = CHR_COMMA, _
            Optional ByVal RowDelimiter As String, _
            Optional ByVal quote As String = CHR_QUOTE, _
            Optional ByVal IgnoreFirstRow As Boolean, _
            Optional ByVal DuckType As Boolean _
            ) As BetterArray
        Dim RawData     As String
        Dim LineEnding  As String
        RawData = ReadStringFromFile(Path)
        If RowDelimiter = vbNullString Then
            LineEnding = DetectLineEndings(RawData)
        Else
            LineEnding = RowDelimiter
        End If
        Set FromCSVFile = FromCSVString( _
                CSVString:=RawData, _
                ColumnDelimiter:=ColumnDelimiter, _
                RowDelimiter:=LineEnding, _
                quote:=quote, _
                IgnoreFirstRow:=IgnoreFirstRow, _
                DuckType:=DuckType _
                )
    End Function
    Public Function FromCSVString( _                                                            ...
            ByVal CSVString As String, _
            Optional ByVal ColumnDelimiter As String = CHR_COMMA, _
            Optional ByVal RowDelimiter As String, _
            Optional ByVal quote As String = CHR_QUOTE, _
            Optional ByVal IgnoreFirstRow As Boolean, _
            Optional ByVal DuckType As Boolean _
            ) As BetterArray
        Dim LocalType   As ArrayTypes
        Dim Parsed()    As Variant
        Dim LineEnding  As String
        If RowDelimiter = vbNullString Then
            LineEnding = DetectLineEndings(CSVString)
        Else
            LineEnding = RowDelimiter
        End If
        If This.ArrayType = BA_MULTIDIMENSION Then
            LocalType = BA_MULTIDIMENSION
        Else
            LocalType = BA_JAGGED
        End If
        Parsed = ParseCSV( _
                Expression:=CSVString, _
                ColumnDelimiter:=ColumnDelimiter, _
                RowDelimiter:=LineEnding, _
                quote:=quote, _
                IgnoreFirstRow:=IgnoreFirstRow, _
```

```
                ReturnJagged:=IIf(LocalType = BA_JAGGED, True, False), _
                Base:=This.LowerBound, _
                DuckType:=DuckType _
                )
        This.Length = GetArrayLength(Parsed)
        This.Capacity = UBound(Parsed)
        This.ArrayType = LocalType
        This.Items = Parsed
        Set FromCSVString = Me
    End Function
    Public Function ToCSVString( _                                                                                          ...
            Optional ByRef Headers As Variant, _
            Optional ByVal ColumnDelimiter As String = CHR_COMMA, _
            Optional ByVal RowDelimiter As String = vbCrLf, _
            Optional ByVal quote As String = CHR_QUOTE, _
            Optional ByVal EncloseAllInQuotes As Boolean, _
            Optional ByVal DateFormat As String, _
            Optional ByVal NumberFormat As String _
            ) As String
        Dim LocalItems() As Variant
        Dim Depth       As Long
        Dim EncodedRecords() As String
        Dim Result      As String
        Dim HeadersArray() As Variant
        LocalItems = InternalItems
        Depth = GetJaggedArrayDepth(LocalItems)
        If Depth > 0 Then
            If Depth = 1 Then
                LocalItems = ConvertOneDimensionArrayToJagged(LocalItems)
            End If
            If Not IsMissing(Headers) Then
                If IsArray(Headers) Then
                    HeadersArray = Array(Headers)
                    LocalItems = InternalConcat(HeadersArray, LocalItems)
                End If
            End If
            EncodedRecords = EncodeCSVRecords(LocalItems, ColumnDelimiter, RowDelimiter,  _
            EncloseAllInQuotes, DateFormat, NumberFormat)
            Result = BuildCSVString(EncodedRecords, ColumnDelimiter, RowDelimiter)
        End If
        ToCSVString = Result
    End Function
    Public Function ToCSVFile( _                                                                                            ...
            ByVal Path As String, _
            Optional ByRef Headers As Variant, _
            Optional ByVal ColumnDelimiter As String = CHR_COMMA, _
            Optional ByVal RowDelimiter As String = vbCrLf, _
            Optional ByVal quote As String = CHR_QUOTE, _
            Optional ByVal EncloseAllInQuotes As Boolean, _
            Optional ByVal DateFormat As String, _
            Optional ByVal NumberFormat As String _
            ) As String
        Dim Content     As String
        If IsMissing(Headers) Then
            Content = Me.ToCSVString( _
                    ColumnDelimiter:=ColumnDelimiter, _
                    RowDelimiter:=RowDelimiter, _
                    quote:=quote, _
                    EncloseAllInQuotes:=EncloseAllInQuotes, _
                    DateFormat:=DateFormat, _
                    NumberFormat:=NumberFormat _
                    )
        Else
            Content = Me.ToCSVString( _
                    Headers, _
                    ColumnDelimiter, _
                    RowDelimiter, _
                    quote, _
                    EncloseAllInQuotes, _
                    DateFormat, _
                    NumberFormat _
                    )
        End If
        PrintStringToFile Path, Content
        ToCSVFile = Content
    End Function
    Private Function DetectLineEndings(ByVal Source As String) As String                                    ...
        If VBA.Strings.InStr(Source, vbCrLf) Then
            DetectLineEndings = vbCrLf
            Exit Function
        End If
        If VBA.Strings.InStr(Source, vbLf) Then
            DetectLineEndings = vbLf
            Exit Function
        End If
        If VBA.Strings.InStr(Source, vbCr) Then
            DetectLineEndings = vbCr
            Exit Function
        End If
        DetectLineEndings = vbCrLf
    End Function
    Private Function ReadStringFromFile(ByVal Path As String) As String                                     ...
        Dim File        As Long
        Dim ByteCount   As Long
        Dim Result      As String
        If Dir(Path) <> vbNullString Then
            File = FreeFile
            Open Path For Input Access Read Lock Read As File
            ByteCount = LOF(File)
            Result = Input$(ByteCount, File)
            Close File
        End If
        ReadStringFromFile = Result
    End Function
    Private Sub PrintStringToFile(ByVal Path As String, ByVal Content As String)                            ...
        Dim File        As Long
        File = FreeFile
        Open Path For Output Access Write As File
        Print #File, Content
        Close File
    End Sub
    Private Function WrapQuote(Optional ByVal Source As String = vbNullString) As String                    ...
        WrapQuote = CHR_QUOTE & Source & CHR_QUOTE
    End Function
    Private Function BuildCSVString( _                                                                      ...
            ByRef records() As String, _
            ByVal ColumnDelimiter As String, _
            ByVal RowDelimiter As String _
            ) As String
        Dim i           As Long
        Dim j           As Long
        Dim lastRow     As Long
        Dim lastCol     As Long
        Dim ValidRow    As Boolean
        lastRow = UBound(records)
        lastCol = UBound(records, 2)
        StringBuilder NewString:=True
        For i = LBound(records) To UBound(records)
            ValidRow = False
            For j = LBound(records, 2) To UBound(records, 2)
                If records(i, j) <> vbNullString Then
                    ValidRow = True
                    Exit For
                End If
            Next
            If ValidRow Then
                For j = LBound(records, 2) To UBound(records, 2)
                    StringBuilder records(i, j)
                    If j = lastCol Then
                        If i <> lastRow Then
                            StringBuilder RowDelimiter
                        End If
                    Else
                        StringBuilder ColumnDelimiter
                    End If
                Next
            End If
        Next
        BuildCSVString = StringBuilder(Final:=True)
    End Function
    Private Function EncodeCSVRecords( _                                                                    ...
            ByRef records() As Variant, _
            ByVal ColumnDelimiter As String, _
            ByVal RowDelimiter As String, _
            ByVal EncloseAllInQuotes As Boolean, _
            ByVal DateFormat As String, _
            ByVal NumberFormat As String _
            ) As String()
        Dim FirstDimBounds() As Long
        Dim SecondDimBounds() As Long
        Dim i            As Long
        Dim j            As Long
        Dim Result()     As String
        Dim CurrentField As Variant
        FirstDimBounds = GetArrayBounds(records)
        SecondDimBounds = GetMaxBoundsAtDimension(records, 2)
        ReDim Result( _
                FirstDimBounds(0) To FirstDimBounds(1), _
                SecondDimBounds(0) To SecondDimBounds(1) _
                )
        For i = FirstDimBounds(0) To FirstDimBounds(1)
            If IsArray(records(i)) Then
                For j = LBound(records(i)) To UBound(records(i))
                    CurrentField = GetScalarRepresentation(records(i)(j))
                    Result(i, j) = EncodeCSVField( _
                            CurrentField, _
                            ColumnDelimiter, _
                            RowDelimiter, _
                            EncloseAllInQuotes, _
                            DateFormat, _
                            NumberFormat _
                            )
                Next
            End If
        Next
        EncodeCSVRecords = Result
    End Function
    Private Function EncodeCSVField( _                                                                      ...
            ByVal Field As Variant, _
            ByVal ColumnDelimiter As String, _
            ByVal RowDelimiter As String, _
            ByVal EncloseAllInQuotes As Boolean, _
            ByVal DateFormat As String, _
            ByVal NumberFormat As String _
            ) As String
        Dim LocalField  As String
        If IsDate(Field) And DateFormat <> vbNullString Then
            On Error Resume Next
            LocalField = Format$(Field, DateFormat)
            On Error GoTo 0
        ElseIf IsNumeric(Field) And NumberFormat <> vbNullString Then
            On Error Resume Next
            LocalField = Format$(Field, NumberFormat)
            On Error GoTo 0
        End If
        If LocalField = vbNullString Then
            LocalField = CStr(Field)
        End If
        LocalField = EscapeCharInString(LocalField, CHR_QUOTE, CHR_QUOTE)
        If EncloseAllInQuotes Or _
                VBA.Strings.InStr(LocalField, RowDelimiter) Or _
                VBA.Strings.InStr(LocalField, ColumnDelimiter) Then
            LocalField = WrapQuote(LocalField)
        End If
        EncodeCSVField = LocalField
        Exit Function
```

```
End Function
Private Function EscapeCharInString( _                                                               ...
        ByVal Destination As String, _
        ByVal Target As String, _
        ByVal Escape As String, _
        Optional ByVal BothSides As Boolean _
        ) As String
    Dim index        As Long
    Dim TargetLength As Long
    Dim EscapeLength As String
    Dim Result       As String
    TargetLength = Len(Target)
    EscapeLength = Len(Escape)
    Result = Destination
    index = 1
    Do
        index = VBA.Strings.InStr(index, Destination, Target)
        If index Then
            Result = InsertIntoStringAtIndex(Result, Escape, index)
            inc index, EscapeLength + TargetLength
            If BothSides Then
                Result = InsertIntoStringAtIndex(Result, Escape, index)
            End If
        End If
    Loop While index > 0
    EscapeCharInString = Result
End Function
Private Function InsertIntoStringAtIndex( _                                                          ...
        ByVal Destination As String, _
        ByVal Source As String, _
        ByVal index As Long _
        ) As String
    Dim Front        As String
    Dim Back         As String
    If index > 1 Then
        Front = Left$(Destination, index)
    End If
    If index < Len(Destination) Then
        Back = Right$(Destination, Len(Destination) - index)
    End If
    InsertIntoStringAtIndex = Front & Source & Back
End Function
Private Function InternalConcat(ParamArray Args() As Variant) As Variant()                           ...
    Dim first()      As Variant
    first = Array()
    InternalConcat = ConcatDelegate(first, Args)
End Function
Private Function ConcatDelegate(ByRef first() As Variant, ParamArray Args() As Variant) As Variant() ...
    Dim i            As Long
    Dim Cursor       As Long
    Dim NewLength    As Long
    Dim Rest()       As Variant
    Dim Result()     As Variant
    Dim Current()    As Variant
    Rest = Args(0)
    Cursor = This.LowerBound
    If IsArrayAllocated(first) Then
        Cursor = LBound(first)
    ElseIf IsArray(Rest(LBound(Rest))) Then
        If IsArrayAllocated(Rest(LBound(Rest))) Then
            Cursor = LBound(Rest(LBound(Rest)))
        End If
    End If
    NewLength = GetArrayLength(first) + GetTotalLengthOfNestedArrays(Rest)
    ReDim Result(Cursor To Max(Cursor + NewLength - 1, Cursor))
    InsertArrayAtIndex Result, first, Cursor
    For i = LBound(Rest) To UBound(Rest)
        If IsArray(Rest(i)) Then
            Current = Rest(i)
            If IsMultidimensionalArray(Current) Then
                Current = MultiToJagged(Current)
            End If
            InsertArrayAtIndex Result, Current, Cursor
        Else
            LetOrSetElement Result(Cursor), Rest(i)
            inc Cursor
        End If
    Next
    ConcatDelegate = Result
End Function
Private Sub InsertArrayAtIndex(ByRef Destination() As Variant, ByRef Source() As Variant, ByRef  _                    ...
index As Long)
    Dim OFFSET      As Long
    Dim SourceLength As Long
    Dim StartingIndex As Long
    If IsArrayAllocated(Source) Then
        StartingIndex = index
        OFFSET = StartingIndex - LBound(Source)
        SourceLength = GetArrayLength(Source)
        Do While index < StartingIndex + SourceLength
            LetOrSetElement Destination(index), Source(index - OFFSET)
            inc index
        Loop
    End If
End Sub
Private Function GetTotalLengthOfNestedArrays(ByRef Source() As Variant) As Long                                    ...
    Dim i           As Long
    Dim Result      As Long
    Dim Current()   As Variant
    For i = LBound(Source) To UBound(Source)
        If IsArray(Source(i)) Then
            Current = Source(i)
            inc Result, GetArrayLength(Current)
        End If
    Next
    GetTotalLengthOfNestedArrays = Result
End Function
Private Function TrimColumnsMultidimensionArray(ByRef Original() As Variant, ByVal AvailableColumns  _              ...
As Long) As Variant()
    Dim i           As Long
    Dim j           As Long
    Dim Result()    As Variant
    Dim OuterBounds(0 To 1) As Long
    Dim InnerBounds(0 To 1) As Long
    Dim CountColumns As Long
    CountColumns = UBound(Original, 2) - LBound(Original, 2) + 1
    If CountColumns > AvailableColumns Then
        OuterBounds(0) = LBound(Original, 1)
        OuterBounds(1) = UBound(Original, 1)
        InnerBounds(0) = LBound(Original, 2)
        InnerBounds(1) = InnerBounds(0) + AvailableColumns - 1
        ReDim Result(OuterBounds(0) To OuterBounds(1), InnerBounds(0) To InnerBounds(1))
        For i = OuterBounds(0) To OuterBounds(1)
            For j = InnerBounds(0) To InnerBounds(1)
                LetOrSetElement Result(i, j), Original(i, j)
            Next
        Next
    Else
        Result = Original
    End If
    TrimColumnsMultidimensionArray = Result
End Function
Private Function TrimRowsMultidimensionArray(ByRef Original() As Variant, ByVal AvailableRows As  _                 ...
Long) As Variant()
    Dim i           As Long
    Dim j           As Long
    Dim Result()    As Variant
    Dim OuterBounds(0 To 1) As Long
    Dim InnerBounds(0 To 1) As Long
    Dim CountRows   As Long
    CountRows = UBound(Original, 1) - LBound(Original, 1) + 1
    If CountRows > AvailableRows Then
        OuterBounds(0) = LBound(Original, 1)
        OuterBounds(1) = OuterBounds(0) + AvailableRows - 1
```

```
            InnerBounds(0) = LBound(Original, 2)
            InnerBounds(1) = UBound(Original, 2)
            ReDim Result(OuterBounds(0) To OuterBounds(1), InnerBounds(0) To InnerBounds(1))
            For i = OuterBounds(0) To OuterBounds(1)
                For j = InnerBounds(0) To InnerBounds(1)
                    LetOrSetElement Result(i, j), Original(i, j)
                Next
            Next
        Else
            Result = Original
        End If
        TrimRowsMultidimensionArray = Result
    End Function
    Private Function inc(ByRef index As Long, Optional ByVal Value As Long = 1) As Long                    ...
        index = index + Value
        inc = index
    End Function
    Private Function dec(ByRef index As Long, Optional ByVal Value As Long = 1) As Long                    ...
        index = index - Value
        dec = index
    End Function
    Private Function StringFactory(ByRef Expression As String) As tString                                  ...
        Dim Result      As tString
        Result.TEXT = Expression
        Result.Length = Len(Expression)
        Result.ByteLength = LenB(Expression)
        StringFactory = Result
    End Function
    Private Function NextDelimBytePos( _                                                                   ...
            ByRef Expression As tString, _
            ByRef Delimiter As tString, _
            Optional ByRef StartIndex As Long = 1 _
            ) As Long
        Dim Result      As Long
        Result = InStrB(StartIndex, Expression.TEXT, Delimiter.TEXT, vbBinaryCompare)
        Do Until (Result And 1) Or (Result = 0)
            Result = InStrB(Result + 1, Expression.TEXT, Delimiter.TEXT, vbBinaryCompare)
        Loop
        NextDelimBytePos = Result
    End Function
    Private Function GetCSVRows( _                                                                         ...
            ByRef Expr As tString, _
            ByRef RowDelim As tString, _
            ByRef LiteralDelim As tString _
            ) As String()
        Dim RowDelimIndex As Long
        Dim LastRowDelim As Long
        Dim RowIndex    As Long
        Dim BlankTrailingRows As Long
        Dim MaxRows     As Long
        Dim LineEndIndices() As Long
        Dim LiteralDelimIndex As Long
        Dim CountRows   As Long
        Dim i           As Long
        Dim Cursor      As Long
        Dim CSVRows()   As String
        RowDelimIndex = NextDelimBytePos(Expr, RowDelim)
        LastRowDelim = InStrRev(Expr.TEXT, RowDelim.TEXT)
        RowIndex = Expr.Length + 1
        Do While RowIndex - LastRowDelim = RowDelim.Length
            RowIndex = LastRowDelim
            LastRowDelim = InStrRev(Expr.TEXT, RowDelim.TEXT, LastRowDelim)
            inc BlankTrailingRows
        Loop
        RowIndex = Empty
        MaxRows = (Expr.Length - Len(Replace(Expr.TEXT, RowDelim.TEXT, vbNullString))) / RowDelim. _
        Length + 1
        ReDim LineEndIndices(0 To MaxRows)
        LiteralDelimIndex = NextDelimBytePos(Expr, LiteralDelim)
        Do While RowDelimIndex > 0
            If RowDelimIndex + RowDelim.ByteLength <= LiteralDelimIndex Or LiteralDelimIndex = 0 Then
                LineEndIndices(CountRows) = RowDelimIndex
                RowDelimIndex = NextDelimBytePos(Expr, RowDelim, RowDelimIndex + RowDelim.ByteLength)
                inc CountRows
            Else
                LiteralDelimIndex = NextDelimBytePos(Expr, LiteralDelim, LiteralDelimIndex + 2)
                If LiteralDelimIndex Then
                    RowDelimIndex = NextDelimBytePos(Expr, RowDelim, LiteralDelimIndex + 2)
                    If RowDelimIndex Then
                        LiteralDelimIndex = NextDelimBytePos(Expr, LiteralDelim, LiteralDelimIndex + 2)
                    End If
                End If
            End If
        Loop
        LineEndIndices(CountRows) = Expr.ByteLength + 1
        dec CountRows, BlankTrailingRows - 1
        ReDim CSVRows(0 To CountRows - 1)
        Cursor = 1
        For i = 0 To CountRows - 1
            CSVRows(i) = MidB$(Expr.TEXT, Cursor, LineEndIndices(i) - Cursor)
            Cursor = LineEndIndices(i) + RowDelim.ByteLength
        Next
        GetCSVRows = CSVRows
    End Function
    Private Function CountCSVColumns(ByRef FirstRow As String, ByRef LiteralDelim As tString, ByVal  _                 ...
    ColumnDelimiter As String) As Long
        Dim i            As Long
        Dim InQuote      As Boolean
        Dim CountColumns As Long
        Dim Char         As String
        CountColumns = 1
        For i = 0 To Len(FirstRow)
            Char = Mid$(FirstRow, i + 1, 1)
            Select Case Char
                Case LiteralDelim.TEXT
                    InQuote = Not InQuote
                Case ColumnDelimiter
                    If Not InQuote Then
                        inc CountColumns
                    End If
            End Select
        Next
        CountCSVColumns = CountColumns
    End Function
    Private Function ParseCSV( _                                                                                  ...
            ByRef Expression As String, _
            ByVal ColumnDelimiter As String, _
            ByVal RowDelimiter As String, _
            ByRef quote As String, _
            ByVal IgnoreFirstRow As Boolean, _
            ByVal ReturnJagged As Boolean, _
            ByVal Base As Long, _
            ByVal DuckType As Boolean _
            ) As Variant()
        Dim i            As Long
        Dim j            As Long
        Dim Cursor       As Long
        Dim ColumnIndex  As Long
        Dim CountColumns As Long
        Dim CountRows    As Long
        Dim lastRow      As Long
        Dim LastColumn   As Long
        Dim StartRow     As Long
        Dim InQuote      As Boolean
        Dim Buffer       As String
        Dim Char         As String
        Dim Frag         As String
        Dim CSVRows()    As String
        Dim JaggedRow()  As Variant
        Dim results()    As Variant
        Dim Element      As Variant
        Dim Expr         As tString
        Dim RowDelim     As tString
        Dim LiteralDelim As tString
        Expr = StringFactory(Expression)
        RowDelim = StringFactory(RowDelimiter)
        LiteralDelim = StringFactory(quote)
        If Expr.ByteLength = 0 Or RowDelim.ByteLength = 0 Then
            ParseCSV = Array()
            Exit Function
        End If
        CSVRows = GetCSVRows(Expr, RowDelim, LiteralDelim)
        CountColumns = CountCSVColumns(CSVRows(0), LiteralDelim, ColumnDelimiter)
        CountRows = UBound(CSVRows) - LBound(CSVRows) + 1
        If IgnoreFirstRow Then
            StartRow = 1
        Else
            StartRow = 0
        End If
        lastRow = Base + CountRows - 1 - StartRow
        LastColumn = Base + CountColumns - 1
        If ReturnJagged Then
            ReDim JaggedRow(Base To LastColumn)
            ReDim results(Base To lastRow)
        Else
            ReDim results(Base To lastRow, Base To LastColumn)
        End If
        For i = StartRow To lastRow
            ColumnIndex = Base
            Buffer = CSVRows(i)
            Cursor = 1
            For j = 0 To Len(Buffer)
                Char = Mid$(Buffer, j + 1, 1)
                If Char = LiteralDelim.TEXT Then
                    InQuote = Not InQuote
                ElseIf Char = ColumnDelimiter Or j = Len(Buffer) Then
                    If Not InQuote Then
                        Frag = Mid$(Buffer, Cursor, j - Cursor + 1)
                        If DuckType Then
                            Element = DuckTypeElement(Frag)
                        Else
                            Element = UnquoteString(Frag)
                        End If
                        If ReturnJagged Then
                            JaggedRow(ColumnIndex) = Element
                        Else
                            results(i - StartRow, ColumnIndex) = Element
                        End If
                        Cursor = j + 2
                        inc ColumnIndex
                        Frag = vbNullString
                    End If
                End If
            Next
            If ReturnJagged Then
                results(i - StartRow) = JaggedRow
            End If
        Next
        ParseCSV = results
    End Function
    Private Function RecursiveFill( _                                                        ...
            ByRef SourceArray() As Variant, _
            ByVal Value As Variant, _
            Optional ByVal StartIndex As Long = MISSING_LONG, _
            Optional ByVal EndIndex As Long = MISSING_LONG _
            ) As Variant()
        Dim RelativeStart As Long
        Dim RelativeEnd As Long
        Dim i           As Long
        If StartIndex = MISSING_LONG Then
            RelativeStart = LBound(SourceArray)
        Else
            RelativeStart = StartIndex
        End If
        If EndIndex = MISSING_LONG Then
            RelativeEnd = UBound(SourceArray)
        Else
            RelativeEnd = EndIndex
        End If
        For i = RelativeStart To RelativeEnd
            If IsArray(SourceArray(i)) Then
                Dim PassThru() As Variant
                PassThru = RecursiveFill(PassThru, Value)
            Else
                LetOrSetElement SourceArray(i), Value
            End If
        Next
        RecursiveFill = SourceArray
    End Function
    Private Function RecursiveEvery( _                                                                    ...
            ByVal SearchElement As Variant, _
            ByRef SearchArray() As Variant, _
            Optional ByVal FromIndex As Long, _
            Optional ByVal CompareTypeNames As Boolean _
            ) As Boolean
        Dim LocalLowerBound As Long
        Dim LocalUpperBound As Long
        Dim i               As Long
        LocalLowerBound = LBound(SearchArray)
        If FromIndex > LocalLowerBound Then LocalLowerBound = FromIndex
        LocalUpperBound = UBound(SearchArray)
        For i = LocalLowerBound To LocalUpperBound
            If IsArray(SearchArray(i)) Then
                Dim PassThru() As Variant
                PassThru = SearchArray(i)
                If Not RecursiveEvery(SearchElement, PassThru, CompareTypeNames:=CompareTypeNames) Then
                    Exit Function
                End If
            ElseIf CompareTypeNames Then
                If Not InStr( _
                        UCase$(TypeName(SearchArray(i))), _
                        UCase$(CStr(SearchElement)) _
                        ) > 0 Then
                    Exit Function
                End If
            Else
                If Not ElementsAreEqual(SearchElement, SearchArray(i)) Then
                    Exit Function
                End If
            End If
        Next
        RecursiveEvery = True
    End Function
    Private Function RecursiveShuffle( _                                                                  ...
            ByRef SourceArray() As Variant, _
            Optional ByVal Recurse As Boolean _
            ) As Variant()
        Dim i           As Long
        Dim j           As Long
        Dim Lower       As Long
        Dim Upper       As Long
        Dim Nested()    As Variant
        Lower = LBound(SourceArray)
        Upper = UBound(SourceArray)
        Randomize
        For i = Upper To Lower + 1 Step -1
            If IsArray(SourceArray(i)) And Recurse Then
                Nested = SourceArray(i)
                SourceArray(i) = RecursiveShuffle(Nested, Recurse)
            End If
            j = Int(Rnd * (i - Lower) + 1)
            If IsArray(SourceArray(j)) And Recurse Then
                Nested = SourceArray(j)
                SourceArray(j) = RecursiveShuffle(Nested, Recurse)
            End If
            Swap SourceArray, i, j
        Next
        RecursiveShuffle = SourceArray
```

```
End Function
Private Function RecursiveReverse( _                                              ...
        ByRef SourceArray() As Variant, _
        Optional ByVal Recurse As Boolean _
        ) As Variant()
    Dim LocalLength As Long
    Dim Middle      As Long
    Dim Lower       As Long
    Dim Upper       As Long
    Dim PassThruArray() As Variant
    LocalLength = GetArrayLength(SourceArray)
    Lower = LBound(SourceArray)
    Middle = Int(LocalLength / 2) + Lower
    Do While Lower <> Middle
        Upper = UBound(SourceArray) + LBound(SourceArray) - Lower
        If IsArray(SourceArray(Lower)) And Recurse Then
            PassThruArray = SourceArray(Lower)
            SourceArray(Lower) = RecursiveReverse(PassThruArray, Recurse)
        End If
        If IsArray(SourceArray(Upper)) And Recurse Then
            PassThruArray = SourceArray(Upper)
            SourceArray(Upper) = RecursiveReverse(PassThruArray, Recurse)
        End If
        Swap SourceArray, Lower, Upper
        inc Lower
    Loop
    RecursiveReverse = SourceArray
End Function
Private Function GetArrayLength(ByRef SourceArray() As Variant) As Long           ...
    Dim Result      As Long
    Result = 0
    If IsArrayAllocated(SourceArray) Then
        If Not IsArrayEmpty(SourceArray) Then
            Result = UBound(SourceArray) - LBound(SourceArray) + 1
        End If
    End If
    GetArrayLength = Result
End Function
Private Sub RecursiveFlatten( _                                                   ...
        ByRef SourceArray() As Variant, _
        ByRef results As BetterArray _
        )
    Dim Element     As Variant
    For Each Element In SourceArray
        If IsArray(Element) Then
            Dim ArrayElement() As Variant
            ArrayElement = Element
            RecursiveFlatten ArrayElement, results
        Else
            results.Push Element
        End If
    Next
End Sub
Private Function RecursiveMax(ByRef SourceArray() As Variant) As Variant          ...
    Dim i           As Long
    Dim Result      As Variant
    If IsMultidimensionalArray(SourceArray) Then
        RaiseError EC_INVALID_MULTIDIMENSIONAL_ARRAY_OPERATION, "Max", "Args"
    Else
        For i = LBound(SourceArray) To UBound(SourceArray)
            If IsArray(SourceArray(i)) Then
                Dim NestedArray() As Variant
                Dim NestedResult As Variant
                NestedArray = SourceArray(i)
                NestedResult = RecursiveMax(NestedArray)
                If IsEmpty(Result) Then
                    Result = NestedResult
                Else
                    If NestedResult > Result Then Result = NestedResult
                End If
            Else
                Dim CurrentValue As Variant
                CurrentValue = GetScalarRepresentation(SourceArray(i))
                If CurrentValue <> OBJECT_REPR Then
                    If IsEmpty(Result) Then
                        Result = CurrentValue
                    Else
                        If CurrentValue > Result Then Result = CurrentValue
                    End If
                End If
            End If
        Next
    End If
    RecursiveMax = Result
End Function
Private Function RecursiveMin(ByRef SourceArray() As Variant) As Variant                                                                              ...
    Dim i           As Long
    Dim Result      As Variant
    If IsMultidimensionalArray(SourceArray) Then
        RaiseError EC_INVALID_MULTIDIMENSIONAL_ARRAY_OPERATION, "Max", "Args"
    Else
        For i = LBound(SourceArray) To UBound(SourceArray)
            If IsArray(SourceArray(i)) Then
                Dim NestedArray() As Variant
                Dim NestedResult As Variant
                NestedArray = SourceArray(i)
                NestedResult = RecursiveMin(NestedArray)
                If IsEmpty(Result) Then
                    Result = NestedResult
                Else
                    If NestedResult < Result Then Result = NestedResult
                End If
            Else
                Dim CurrentValue As Variant
                CurrentValue = GetScalarRepresentation(SourceArray(i))
                If CurrentValue <> OBJECT_REPR Then
                    If IsEmpty(Result) Then
                        Result = CurrentValue
                    Else
                        If CurrentValue < Result Then Result = CurrentValue
                    End If
                End If
            End If
        Next
    End If
    RecursiveMin = Result
End Function
Private Function RecursiveIncludes( _                                                                                                                ...
        ByVal SearchElement As Variant, _
        ByRef SearchArray() As Variant, _
        Optional ByVal FromIndex As Long, _
        Optional ByVal CompareTypeNames As Boolean, _
        Optional ByVal Recurse As Boolean _
        ) As Boolean
    Dim LocalLowerBound As Long
    Dim LocalUpperBound As Long
    Dim i               As Long
    LocalLowerBound = LBound(SearchArray)
    If FromIndex > LocalLowerBound Then LocalLowerBound = FromIndex
    LocalUpperBound = UBound(SearchArray)
    For i = LocalLowerBound To LocalUpperBound
        If CompareTypeNames Then
            If InStr( _
                    UCase$(TypeName(SearchArray(i))), _
                    UCase$(CStr(SearchElement)) _
                    ) > 0 _
                    Or _
                    UCase$(CStr(SearchElement)) = "OBJECT" And _
                    IsObject(SearchArray(i)) _
                    Then
                RecursiveIncludes = True
                Exit Function
            End If
        End If
```

```
            If IsArray(SearchArray(i)) And Recurse Then
                Dim PassThru() As Variant
                PassThru = SearchArray(i)
                If RecursiveIncludes( _
                        SearchElement, _
                        PassThru, _
                        CompareTypeNames:=CompareTypeNames, _
                        Recurse:=Recurse _
                        ) Then
                    RecursiveIncludes = True
                    Exit Function
                End If
            ElseIf Not CompareTypeNames Then
                If ElementsAreEqual(SearchElement, SearchArray(i)) Then
                    RecursiveIncludes = True
                    Exit Function
                End If
            End If
        Next
        RecursiveIncludes = False
    End Function
    Private Function RecursiveFilter( _                                                                   ...
            ByRef SourceArray() As Variant, _
            ByVal Match As Variant, _
            ByVal Include As Boolean, _
            ByVal Recurse As Boolean, _
            Optional ByVal CompareTypeNames As Boolean _
            ) As Variant()
        Dim Result      As BetterArray
        Dim LocalLowerBound As Long
        Dim LocalUpperBound As Long
        Dim i           As Long
        Dim IsMatch     As Boolean
        LocalLowerBound = LBound(SourceArray)
        LocalUpperBound = UBound(SourceArray)
        Set Result = New BetterArray
        Result.LowerBound = This.LowerBound
        For i = LocalLowerBound To LocalUpperBound
            If IsArray(SourceArray(i)) And Recurse Then
                If IsArrayAllocated(SourceArray(i)) Then
                    Dim LocalItems() As Variant
                    LocalItems = SourceArray(i)
                    LocalItems = RecursiveFilter( _
                            LocalItems, _
                            Match, _
                            Include, _
                            Recurse, _
                            CompareTypeNames _
                            )
                    If Not RecursiveEvery(Empty, LocalItems) Then
                        Result.Push LocalItems
                    End If
                Else
                    If CompareTypeNames Then
                        IsMatch = (InStr( _
                                UCase$(TypeName(SourceArray(i))), _
                                UCase$(CStr(Match)) _
                                ) > 0)
                        If (Include And IsMatch) Or (Not Include And Not IsMatch) Then
                            Result.Push GetEmptyArray
                        End If
                    Else
                        If Not Include Then
                            Result.Push GetEmptyArray
                        End If
                    End If
                End If
            Else
                If CompareTypeNames Then
                    IsMatch = (InStr( _
                            UCase$(TypeName(SourceArray(i))), _
                            UCase$(CStr(Match)) _
                            ) > 0)
                Else
                    IsMatch = ElementsAreEqual(Match, SourceArray(i))
                End If
                If (Include And IsMatch) Or (Not Include And Not IsMatch) Then
                    Result.Push SourceArray(i)
                End If
            End If
        Next
        RecursiveFilter = Result.Items
    End Function
    Private Function ElementsAreEqual( _                                          ...
            ByVal Expected As Variant, _
            ByVal Actual As Variant _
            ) As Boolean
        Const Epsilon   As Double = 0.0000000000001
        Dim Result      As Boolean
        Dim i           As Long
        On Error GoTo ErrHandler
        If IsArray(Expected) Or IsArray(Actual) Then
            If IsArray(Expected) And IsArray(Actual) Then
                If LBound(Expected) = LBound(Actual) And _
                        UBound(Expected) = UBound(Actual) Then
                    Dim CurrentlyEqual As Boolean
                    CurrentlyEqual = True
                    For i = LBound(Expected) To UBound(Actual)
                        If Not ElementsAreEqual(Expected(i), Actual(i)) Then
                            CurrentlyEqual = False
                            Exit For
                        End If
                    Next
                    Result = CurrentlyEqual
                End If
            End If
        ElseIf IsEmpty(Expected) Or IsEmpty(Actual) Then
            If IsEmpty(Expected) And IsEmpty(Actual) Then Result = True
        ElseIf IsObject(Expected) Or IsObject(Actual) Then
            If IsObject(Expected) And IsObject(Actual) Then
                If Expected Is Actual Then Result = True
            End If
        ElseIf IsNumeric(Expected) Or IsNumeric(Actual) Then
            If IsNumeric(Expected) And IsNumeric(Actual) Then
                Dim Diff As Double
                Diff = Abs(Expected - Actual)
                If Diff <= (IIf( _
                        Abs(Expected) < Abs(Actual), _
                        Abs(Actual), _
                        Abs(Expected) _
                        ) * Epsilon) Then
                    Result = True
                End If
            End If
        ElseIf Expected = Actual Then
            Result = True
        End If
        ElementsAreEqual = Result
        Exit Function
    ErrHandler:
        ElementsAreEqual = False
    End Function
    Private Function ParseDelimitedArrayString( _                                 ...
            ByVal SourceString As String, _
            ByVal ValueSeparator As String, _
            ByVal Opener As String, _
            ByVal Closer As String, _
            Optional ByRef Cursor As Long = 2 _
            ) As Variant()
        Dim CurrentChar As String
        Dim LocalResult() As Variant
        Dim i           As Long
        Dim BreakLoop   As Boolean
        Dim NextOpener  As Long
        Dim NextCloser  As Long
        CurrentChar = Mid$(SourceString, Cursor, 1)
        NextOpener = InStr(Cursor, SourceString, Opener)
        NextCloser = InStr(Cursor, SourceString, Closer)
        If CurrentChar <> Opener And (NextCloser < NextOpener Or NextOpener = 0) Then
            ParseDelimitedArrayString = ParseArraySegmentFromString(SourceString, Cursor, NextCloser,  _
            ValueSeparator)
            Exit Function
        End If
        i = This.LowerBound
        Do
            If CurrentChar = Opener Then
                If i > This.LowerBound Then
                    ReDim Preserve LocalResult(i)
                Else
                    LocalResult = GetEmptyArray
                End If
                inc Cursor
                LocalResult(i) = ParseDelimitedArrayString( _
                        SourceString, _
                        ValueSeparator, _
                        Opener, _
                        Closer, _
                        Cursor _
                        )
                inc i
            End If
            CurrentChar = Mid$(SourceString, Cursor, 1)
            Do Until CurrentChar = Opener
                inc Cursor
                CurrentChar = Mid$(SourceString, Cursor, 1)
                If CurrentChar = Closer Or Cursor >= Len(SourceString) Then
                    BreakLoop = True
                    Exit Do
                End If
            Loop
        Loop Until BreakLoop
        ParseDelimitedArrayString = LocalResult
    End Function
    Private Function ParseArraySegmentFromString( _                                                    ...
            ByVal SourceString As String, _
            ByRef Cursor As Long, _
            ByVal NextCloser As Long, _
            ByVal ValueSeparator As String _
            ) As Variant()
        Dim SegmentLength As Long
        Dim Segment()   As String
        SegmentLength = IIf(NextCloser = 0, Len(SourceString), NextCloser - Cursor)
        Segment = Split(Mid$(SourceString, Cursor, SegmentLength), ValueSeparator)
        Cursor = NextCloser
        ParseArraySegmentFromString = DuckTypeStringArray(Segment)
    End Function
    Private Function UnquoteString(ByRef Element As String) As String                                  ...
        Dim LocalElement As String
        LocalElement = Trim$(Element)
        If LocalElement = vbNullString Then
            UnquoteString = vbNullString
        Else
            If Asc(Left$(LocalElement, 1)) = 34 And Asc(Right$(LocalElement, 1)) = 34 Then
                LocalElement = Mid$(LocalElement, 2, Len(LocalElement) - 2)
            End If
            UnquoteString = Trim$(LocalElement)
        End If
    End Function
    Private Function DuckTypeElement(ByRef Element As String) As Variant                               ...
        Dim Result      As Variant
        Dim LocalElement As String
        LocalElement = Element
        If UCase$(LocalElement) = "TRUE" Or UCase$(Element) = "FALSE" Then
            Result = CBool(LocalElement)
        ElseIf IsNumeric(LocalElement) Then
            Const Epsilon As Double = 2 ^ -52
```

```
            Dim Diff    As Double
            Diff = Abs(Fix(LocalElement) - LocalElement)
            If Diff > Epsilon Then
                Result = CDbl(LocalElement)
            Else
                Result = CLng(LocalElement)
            End If
        Else
            Result = UnquoteString(LocalElement)
        End If
        DuckTypeElement = Result
    End Function
    Private Function DuckTypeStringArray(ByRef SourceArray() As String) As Variant()                    ...
        Dim i            As Long
        Dim LocalLowerBound As Long
        Dim LocalUpperBound As Long
        Dim Result()     As Variant
        LocalLowerBound = LBound(SourceArray)
        LocalUpperBound = UBound(SourceArray)
        ReDim Result(LocalLowerBound To LocalUpperBound)
        For i = LocalLowerBound To LocalUpperBound
            Result(i) = DuckTypeElement(SourceArray(i))
        Next
        DuckTypeStringArray = Result
    End Function
    Private Sub PopulateErrorDefinitions()                                                              ...
        Dim Source      As String
        Source = TypeName(Me)
        This.ErrorDefinitions(ErrorCodes.EC_EXPECTED_RANGE_OBJECT) = ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_EXPECTED_RANGE_OBJECT, _
                Source:=Source, _
                Description:="Range Object Expected" _
                )
        This.ErrorDefinitions(ErrorCodes.EC_MAX_DIMENSIONS_LIMIT) = ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_MAX_DIMENSIONS_LIMIT, _
                Source:=Source, _
                Description:="Cannot convert structure of arrays with more than 20 dimensions." _
                )
        This.ErrorDefinitions(ErrorCodes.EC_EXPECTED_COLLECTION_OBJECT) = ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_EXPECTED_COLLECTION_OBJECT, _
                Source:=Source, _
                Description:="Valid Collection Object Expected" _
                )
        This.ErrorDefinitions(ErrorCodes.EC_EXCEEDS_MAX_SORT_DEPTH) = ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_EXCEEDS_MAX_SORT_DEPTH, _
                Source:=Source, _
                Description:="Cannot sort on arrays with more than 2 dimensions" _
                )
        This.ErrorDefinitions(ErrorCodes.EC_EXPECTED_JAGGED_ARRAY) = ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_EXPECTED_JAGGED_ARRAY, _
                Source:=Source, _
                Description:="Expected jagged array." _
                )
        This.ErrorDefinitions(ErrorCodes.EC_EXPECTED_MULTIDIMENSION_ARRAY) = ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_EXPECTED_MULTIDIMENSION_ARRAY, _
                Source:=Source, _
                Description:="Expected multidimension array." _
                )
        This.ErrorDefinitions(ErrorCodes.EC_EXPECTED_ARRAY) = ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_EXPECTED_ARRAY, _
                Source:=Source, _
                Description:="Expected array." _
                )
        This.ErrorDefinitions(ErrorCodes.EC_NULL_STRING) = ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_NULL_STRING, _
                Source:=Source, _
                Description:="Cannot parse from a null string. Expected string with length greater than  _
                0." _
                )
        This.ErrorDefinitions(ErrorCodes.EC_UNALLOCATED_ARRAY) = ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_UNALLOCATED_ARRAY, _
                Source:=Source, _
                Description:="Cannot operate on unallocated array." _
                )
        This.ErrorDefinitions(ErrorCodes.EC_UNDEFINED_ARRAY) = ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_UNDEFINED_ARRAY, _
                Source:=Source, _
                Description:="Array is undefined." _
                )
        This.ErrorDefinitions(ErrorCodes.EC_INVALID_MULTIDIMENSIONAL_ARRAY_OPERATION) =  _
        ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_INVALID_MULTIDIMENSIONAL_ARRAY_OPERATION, _
                Source:=Source, _
                Description:="Unable to perform the requested operation on a multidimensional array." _
                )
        This.ErrorDefinitions(ErrorCodes.EC_EXPECTED_VARIANT_ARRAY) = ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_EXPECTED_VARIANT_ARRAY, _
                Source:=Source, _
                Description:="Unable to perform the requested operation on a typed array." _
                )
        This.ErrorDefinitions(ErrorCodes.EC_EXCEEDS_MAX_ARRAY_LENGTH) = ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_EXCEEDS_MAX_ARRAY_LENGTH, _
                Source:=Source, _
                Description:="The requested operation would result in an array which exceeds the  _
                maximum possible length." _
                )
        This.ErrorDefinitions(ErrorCodes.EC_STRING_TYPE_EXPECTED) = ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_STRING_TYPE_EXPECTED, _
                Source:=Source, _
                Description:="Expected a String or String-coercible type." _
                )
        This.ErrorDefinitions(ErrorCodes.EC_CANNOT_CONVERT_TO_REQUESTED_STRUCTURE) =  _
        ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_CANNOT_CONVERT_TO_REQUESTED_STRUCTURE, _
                Source:=Source, _
                Description:="The stored array cannot be converted to the requested structure." _
                )
        This.ErrorDefinitions(ErrorCodes.EC_CANNOT_SORT_OBJECTS) = ErrorDefinitionFactory( _
                Number:=ErrorCodes.EC_CANNOT_SORT_OBJECTS, _
                Source:=Source, _
                Description:="The array contains Objects which cannot be sorted." _
                )
    End Sub
    Private Function ErrorDefinitionFactory( _                                                              ...
            ByVal Number As Long, _
            ByVal Source As String, _
            ByVal Description As String _
            ) As ErrorDefinition
        Dim Result      As ErrorDefinition
        Result.Number = Number
        Result.Source = Source
        Result.Description = Description
        ErrorDefinitionFactory = Result
    End Function
    Private Sub RaiseError( _                                                                               ...
            ByVal ErrorCode As ErrorCodes, _
            ByVal Caller As String, _
            Optional ByVal ArgName As String _
            )
        Dim CurrentError As ErrorDefinition
        Dim LocalArgName As String
        If ArgName <> vbNullString Then LocalArgName = "Argument " & ArgName & ": "
        CurrentError = This.ErrorDefinitions(ErrorCode)
        Err.Raise CurrentError.Number, _
                CurrentError.Source & "." & Caller, _
                LocalArgName & CurrentError.Description
    End Sub
    Private Function EnsureScalar1DArray(ByRef SourceArray() As Variant) As Variant()                     ...
        Dim i           As Long
        Dim LocalLowerBound As Long
        Dim LocalUpperBound As Long
        Dim Result()    As Variant
        LocalLowerBound = LBound(SourceArray)
        LocalUpperBound = UBound(SourceArray)
        ReDim Result(LocalLowerBound To LocalUpperBound)
        For i = LocalLowerBound To LocalUpperBound
            Result(i) = GetScalarRepresentation(SourceArray(i))
        Next
        EnsureScalar1DArray = Result
    End Function
    Private Function ConvertOneDimensionArrayToJagged(ByRef SourceArray() As Variant) As Variant()        ...
        Dim i           As Long
        Dim LocalUpperBound As Long
        Dim LocalLowerBound As Long
        Dim Result()    As Variant
        LocalUpperBound = UBound(SourceArray)
        LocalLowerBound = LBound(SourceArray)
        ReDim Result(LocalLowerBound To LocalUpperBound)
        For i = LocalLowerBound To LocalUpperBound
            Result(i) = Array(SourceArray(i))
        Next
        ConvertOneDimensionArrayToJagged = Result
    End Function
    Private Function Transpose1DArray(ByRef SourceArray() As Variant) As Variant()        ...
        Dim i           As Long
        Dim j           As Long
        Dim LocalUpperBound As Long
        Dim LocalLowerBound As Long
        Dim Result()    As Variant
        LocalUpperBound = UBound(SourceArray)
        LocalLowerBound = LBound(SourceArray)
        ReDim Result(LocalLowerBound To LocalUpperBound, _
                LocalLowerBound To LocalLowerBound)
        j = LocalLowerBound
        For i = LocalLowerBound To LocalUpperBound
            Result(j, LocalLowerBound) = SourceArray(i)
            inc j
        Next
        Transpose1DArray = Result
    End Function
    Private Function Transpose2DArray(ByRef SourceArray() As Variant) As Variant()        ...
        Dim CurrentRow   As Long
        Dim LowerBoundRow As Long
        Dim UpperBoundRow As Long
        Dim CurrentColumn As Long
        Dim LowerBoundCol As Long
        Dim UpperBoundCol As Long
        Dim Result()    As Variant
        LowerBoundCol = LBound(SourceArray, 1)
        UpperBoundCol = UBound(SourceArray, 1)
        LowerBoundRow = LBound(SourceArray, 2)
        UpperBoundRow = UBound(SourceArray, 2)
        ReDim Result(LowerBoundRow To UpperBoundRow, LowerBoundCol To UpperBoundCol)
        For CurrentRow = LowerBoundRow To UpperBoundRow
            For CurrentColumn = LowerBoundCol To UpperBoundCol
                Result(CurrentRow, CurrentColumn) = SourceArray(CurrentColumn, CurrentRow)
            Next
        Next
        Transpose2DArray = Result
    End Function
    Private Function TransposeArrayOfArrays(ByRef SourceArray() As Variant) As Variant()        ...
        Dim CurrentRow   As Long
        Dim LowerBoundRow As Long
        Dim UpperBoundRow As Long
        Dim CurrentColumn As Long
        Dim LowerBoundCol As Long
        Dim UpperBoundCol As Long
        Dim Result()    As Variant
        Dim NestedBounds() As Long
        Dim Nested()    As Variant
        NestedBounds = GetMaxBoundsAtDimension(SourceArray, 2)
        LowerBoundCol = LBound(SourceArray)
        UpperBoundCol = UBound(SourceArray)
        LowerBoundRow = NestedBounds(0)
        UpperBoundRow = NestedBounds(1)
        If LowerBoundRow = UpperBoundRow Then
```

```
            ReDim Result(LowerBoundCol To UpperBoundCol)
            CurrentRow = LowerBoundRow
            For CurrentColumn = LowerBoundCol To UpperBoundCol
                Result(CurrentColumn) = SourceArray(CurrentColumn)(CurrentRow)
            Next
        ElseIf LowerBoundCol = UpperBoundCol Then
            ReDim Result(LowerBoundRow To UpperBoundRow)
            CurrentColumn = LowerBoundCol
            For CurrentRow = UpperBoundRow To UpperBoundRow
                Result(CurrentRow) = SourceArray(CurrentColumn)(CurrentRow)
            Next
        Else
            ReDim Result(LowerBoundRow To UpperBoundRow)
            For CurrentRow = LowerBoundRow To UpperBoundRow
                ReDim Nested(LowerBoundCol To UpperBoundCol)
                For CurrentColumn = LowerBoundCol To UpperBoundCol
                    Nested(CurrentColumn) = SourceArray(CurrentColumn)(CurrentRow)
                Next
                Result(CurrentRow) = Nested
            Next
        End If
        TransposeArrayOfArrays = Result
    End Function
    Private Function GetEmptyArray() As Variant()                                          ...
        Dim Result()    As Variant
        ReDim Result(This.LowerBound To This.LowerBound)
        GetEmptyArray = Result
    End Function
    Private Sub ApplySortMethod( _                                                         ...
            ByRef SourceArray() As Variant, _
            ByVal LocalArrayType As ArrayTypes, _
            Optional ByVal Col As Long _
            )
        Select Case This.SortMethod
            Case SortMethods.SM_TIMSORT
                TimSort SourceArray, LocalArrayType, Col
            Case SortMethods.SM_QUICKSORT_RECURSIVE
                QuickSortRecursive _
                        SourceArray, _
                        LBound(SourceArray), _
                        UBound(SourceArray), _
                        LocalArrayType, _
                        Col
            Case SortMethods.SM_QUICKSORT_ITERATIVE
                QuickSortIterative _
                        SourceArray, _
                        LBound(SourceArray), _
                        UBound(SourceArray), _
                        LocalArrayType, _
                        Col
        End Select
    End Sub
    Private Function GetComparisonItem( _                                                  ...
            ByRef Source() As Variant, _
            ByVal index As Long, _
            ByVal Col As Long, _
            ByVal LocalArrayType As ArrayTypes _
            ) As Variant
        If IsObject(Source(index)) Then
            GetComparisonItem = ObjPtr(Source(index))
        Else
            If LocalArrayType = BA_JAGGED Then
                GetComparisonItem = Source(index)(Col)
            Else
                GetComparisonItem = Source(index)
            End If
        End If
    End Function
    Private Sub InsertionSort( _                                                           ...
            ByRef Source() As Variant, _
            ByVal LeftIndex As Variant, _
            ByVal RightIndex As Variant, _
            ByVal LocalArrayType As ArrayTypes, _
            Optional ByVal Col As Long _
            )
        Dim i           As Long
        Dim j           As Long
        Dim Pivot       As Variant
        Dim PivotCompVal As Variant
        Dim Current     As Variant
        Dim CurrentCompval As Variant
        For i = LeftIndex + 1 To RightIndex
            LetOrSetElement Pivot, Source(i)
            PivotCompVal = GetComparisonItem(Source, i, Col, LocalArrayType)
            j = i - 1
            Do While j >= LeftIndex
                LetOrSetElement Current, Source(j)
                CurrentCompval = GetComparisonItem(Source, j, Col, LocalArrayType)
                If CurrentCompval > PivotCompVal Then
                    LetOrSetElement Source(j + 1), Current
                    dec j
                Else
                    Exit Do
                End If
            Loop
            LetOrSetElement Source(j + 1), Pivot
        Next
    End Sub
    Private Sub MergeSort( _                                                                        ...
            ByRef Source() As Variant, _
            ByVal StartIndex As Long, _
            ByVal MidIndex As Long, _
            ByVal EndIndex As Long, _
            ByVal LocalArrayType As ArrayTypes, _
            Optional ByVal Col As Long _
            )
        Dim LowLength   As Long
        Dim HighLength  As Long
        Dim i           As Long
        Dim LowIndex    As Long
        Dim HighIndex   As Long
        Dim LowPartition() As Variant
        Dim HighPartition() As Variant
        Dim LowItem     As Variant
        Dim HighItem    As Variant
        LowLength = MidIndex - StartIndex + 1
        HighLength = EndIndex - MidIndex
        ReDim LowPartition(0 To Max(LowLength - 1, 0))
        ReDim HighPartition(0 To Max(HighLength - 1, 0))
        i = 0
        Do While i < LowLength
            LetOrSetElement LowPartition(i), Source(StartIndex + i)
            inc i
        Loop
        i = 0
        Do While i < HighLength
            LetOrSetElement HighPartition(i), Source(MidIndex + 1 + i)
            inc i
        Loop
        i = StartIndex
        Do While LowIndex < LowLength And HighIndex < HighLength
            LowItem = GetComparisonItem(LowPartition, LowIndex, Col, LocalArrayType)
            HighItem = GetComparisonItem(HighPartition, HighIndex, Col, LocalArrayType)
            If LowItem <= HighItem Then
                LetOrSetElement Source(i), LowPartition(LowIndex)
                inc LowIndex
            Else
                LetOrSetElement Source(i), HighPartition(HighIndex)
                inc HighIndex
            End If
            inc i
        Loop
        Do While LowIndex < LowLength
            LetOrSetElement Source(i), LowPartition(LowIndex)
```

```
            inc LowIndex
            inc i
        Loop
        Do While HighIndex < HighLength
            LetOrSetElement Source(i), HighPartition(HighIndex)
            inc HighIndex
            inc i
        Loop
    End Sub
    Private Sub TimSort( _                                                                                    ...
            ByRef Source() As Variant, _
            ByVal LocalArrayType As ArrayTypes, _
            Optional ByVal Col As Long _
            )
        Const MIN_RUN   As Long = 32
        Dim i           As Long
        Dim SourceLength As Long
        SourceLength = GetArrayLength(Source)
        For i = LBound(Source) To UBound(Source) Step MIN_RUN
            InsertionSort Source, i, Min((i + MIN_RUN - 1), UBound(Source)), LocalArrayType, Col
        Next
        Dim Size        As Long
        Dim Start       As Long
        Dim Midpoint    As Long
        Dim Endpoint    As Long
        Dim LastIndex   As Long
        LastIndex = UBound(Source)
        Size = MIN_RUN
        Do While Size < SourceLength
            Start = LBound(Source)
            Do While Start < UBound(Source)
                Midpoint = Min(Start + Size - 1, LastIndex)
                Endpoint = Min(Start + Size * 2 - 1, LastIndex)
                MergeSort Source, Start, Midpoint, Endpoint, LocalArrayType, Col
                inc Start, Size * 2
            Loop
            inc Size, Size
        Loop
    End Sub
    Private Sub QuickSortRecursive( _                                                                          ...
            ByRef SourceArray() As Variant, _
            ByVal Low As Long, _
            ByVal High As Long, _
            ByVal LocalArrayType As ArrayTypes, _
            Optional ByVal Col As Long _
            )
        Dim PartitionIndex As Long
        If Low < High Then
            PartitionIndex = QsPartition(SourceArray, Low, High, LocalArrayType, Col)
            QuickSortRecursive SourceArray, Low, PartitionIndex - 1, LocalArrayType, Col
            QuickSortRecursive SourceArray, PartitionIndex + 1, High, LocalArrayType, Col
        End If
    End Sub
    Private Sub QuickSortIterative( _                                                                          ...
            ByRef SourceArray() As Variant, _
            ByVal Low As Long, _
            ByVal High As Long, _
            ByVal LocalArrayType As ArrayTypes, _
            Optional ByVal Col As Long _
            )
        Dim PartitionIndex As Long
        Dim Stack()     As Long
        Dim Top         As Long
        Dim LocalLow    As Long
        Dim LocalHigh   As Long
        LocalLow = Low
        LocalHigh = High
        ReDim Stack(0 To LocalHigh - LocalLow + 1)
        Top = -1
        Stack(inc(Top)) = LocalLow
        Stack(inc(Top)) = LocalHigh
        Do While Top >= 0
            LocalHigh = Stack(Top)
            LocalLow = Stack(dec(Top))
            dec Top
            PartitionIndex = QsPartition(SourceArray, LocalLow, LocalHigh, LocalArrayType, Col)
            If PartitionIndex - LocalLow > 1 Then
                Stack(inc(Top)) = LocalLow
                Stack(inc(Top)) = PartitionIndex - 1
            End If
            If PartitionIndex + 1 < LocalHigh Then
                Stack(inc(Top)) = PartitionIndex + 1
                Stack(inc(Top)) = LocalHigh
            End If
        Loop
    End Sub
    Private Function QsPartition( _                                                                    ...
            ByRef SourceArray() As Variant, _
            ByVal Low As Long, _
            ByVal High As Long, _
            ByVal LocalArrayType As ArrayTypes, _
            Optional ByVal Col As Long _
            ) As Long
        Dim i           As Long
        Dim j           As Long
        Dim Pivot       As Variant
        Dim Current     As Variant
        Pivot = GetComparisonItem(SourceArray, High, Col, LocalArrayType)
        i = Low - 1
        For j = Low To High - 1
            Current = GetComparisonItem(SourceArray, j, Col, LocalArrayType)
            If Current <= Pivot Then
                inc i
                Swap SourceArray, i, j
            End If
        Next
        Swap SourceArray, i + 1, High
        QsPartition = i + 1
    End Function
    Private Sub Swap(ByRef SourceArray() As Variant, ByVal i As Long, ByVal j As Long)                ...
        Dim Element     As Variant
        LetOrSetElement Element, SourceArray(i)
        LetOrSetElement SourceArray(i), SourceArray(j)
        LetOrSetElement SourceArray(j), Element
    End Sub
    Private Sub LetOrSetElement(ByRef Destination As Variant, ByRef Source As Variant)                ...
        If IsObject(Source) Then
            Set Destination = Source
        Else
            Destination = Source
        End If
    End Sub
    Private Function StringBuilder( _                                                                  ...
            Optional ByVal Fragment As String = vbNullString, _
            Optional ByVal Final As Boolean, _
            Optional ByVal NewString As Boolean _
            ) As String
        Static Buffer() As Byte
        Static BufferLength As Long
        Static BufferCapacity As Long
        Static Cursor   As Long
        Static FirstIndex As Long
        Dim FragLength  As Long
        Dim LastIndex   As Long
        Dim i           As Long
        FragLength = LenB(Fragment)
        If Cursor < 1 Or NewString Then
            Buffer = Fragment
            BufferLength = FragLength
            BufferCapacity = FragLength
            FirstIndex = LBound(Buffer)
            Cursor = UBound(Buffer)
        ElseIf FragLength > 0 Then
            inc BufferLength, FragLength
```

```
            LastIndex = FirstIndex + BufferLength - 1
            If BufferLength > BufferCapacity Then
                Do
                    BufferCapacity = BufferCapacity * 2
                Loop While BufferLength > BufferCapacity
                ReDim Preserve Buffer(FirstIndex To FirstIndex + BufferCapacity)
            End If
            For i = Cursor + 1 To LastIndex
                Buffer(i) = AscB(MidB$(Fragment, i - Cursor, 1))
            Next
            Cursor = LastIndex
        End If
        If Final Then
            ReDim Preserve Buffer(FirstIndex To Cursor)
            StringBuilder = CStr(Buffer)
            Erase Buffer
            Cursor = Empty
        End If
    End Function
    Private Sub RecursiveToString( _                                                                    ...
            ByRef SourceArray() As Variant, _
            ByVal PrettyPrint As Boolean, _
            ByVal Separator As String, _
            ByVal OpeningDelimiter As String, _
            ByVal ClosingDelimiter As String, _
            ByVal QuoteStrings As Boolean, _
            Optional ByVal Tabs As Long = 1 _
            )
        Const TabWidth  As Long = 2
        Dim i           As Long
        Dim Sep         As String
        StringBuilder OpeningDelimiter
        For i = LBound(SourceArray) To UBound(SourceArray)
            Sep = IIf(i = UBound(SourceArray), ClosingDelimiter, Separator)
            If IsArray(SourceArray(i)) Then
                Dim Nested() As Variant
                Nested = SourceArray(i)
                If PrettyPrint Then
                    StringBuilder vbCrLf & Space(TabWidth * Tabs)
                End If
                RecursiveToString Nested, PrettyPrint, Separator, OpeningDelimiter, ClosingDelimiter,  _
                QuoteStrings, Tabs + 1
                If i = UBound(SourceArray) And PrettyPrint Then
                    StringBuilder vbCrLf
                    StringBuilder Space(TabWidth * (Tabs - 1))
                End If
                StringBuilder Sep
            Else
                StringBuilder Replace( _
                        CStr(GetScalarRepresentation(SourceArray(i), QuoteStrings)), _
                        Separator, _
                        vbNullString _
                        ) & Sep
            End If
        Next
    End Sub
    Private Function GetScalarRepresentation( _                                                         ...
            ByRef Element As Variant, _
            Optional ByVal QuoteStrings As Boolean _
            ) As Variant
        Dim Result      As Variant
        If IsObject(Element) Then
            On Error Resume Next
            Result = Element
            On Error GoTo 0
            If IsEmpty(Result) Then
                Result = OBJECT_REPR
            End If
        Else
            Result = Element
        End If
        If IsArray(Result) And Not IsEmpty(Result) Then
```

```vb
        Dim PassThruArray() As Variant
        PassThruArray = Result
        StringBuilder NewString:=True
        RecursiveToString PassThruArray, False, CHR_COMMA, "{", "}", QuoteStrings
        Result = StringBuilder(Final:=True)
    End If
    If Not IsNumeric(Result) And QuoteStrings Then
        GetScalarRepresentation = Chr$(34) & Result & Chr$(34)
    Else
        GetScalarRepresentation = Result
    End If
End Function
Private Function GetArrayType(ByVal SourceArray As Variant) As ArrayTypes                    ...
    Dim Result      As ArrayTypes
    If IsArray(SourceArray) Then
        If Not IsArrayAllocated(SourceArray) Then
            Result = BA_UNALLOCATED
        Else
            If IsMultidimensionalArray(SourceArray) Then
                Result = BA_MULTIDIMENSION
            ElseIf IsJaggedArray(SourceArray) Then
                Result = BA_JAGGED
            Else
                Result = BA_ONEDIMENSION
            End If
        End If
    Else
        If IsEmpty(SourceArray) Then
            Result = BA_UNDEFINED
        Else
            RaiseError EC_EXPECTED_ARRAY, "getArrayType", "sourceArray"
        End If
    End If
    GetArrayType = Result
End Function
Private Function IsArrayEmpty(ByVal SourceArray As Variant) As Boolean                       ...
    Dim Result      As Boolean
    If LBound(SourceArray) = UBound(SourceArray) Then
        On Error Resume
        Result = (SourceArray(LBound(SourceArray)) = Empty)
        On Error GoTo 0
    End If
    IsArrayEmpty = Result
End Function
Private Function IsArrayAllocated(ByVal SourceArray As Variant) As Boolean                   ...
    On Error Resume Next
    IsArrayAllocated = ( _
            IsArray(SourceArray) And _
            Not IsError(LBound(SourceArray, 1)) And _
            LBound(SourceArray, 1) <= UBound(SourceArray, 1) _
            )
    On Error GoTo 0
End Function
Private Function IsJaggedArray(ByVal SourceArray As Variant) As Boolean                      ...
    If IsArray(SourceArray) Then
        On Error GoTo ErrHandler
        Dim Element As Variant
        For Each Element In SourceArray
            If IsArray(Element) Then
                IsJaggedArray = True
                Exit Function
            End If
        Next
        On Error GoTo 0
    End If
    Exit Function
ErrHandler:
    Err.clear
End Function
Private Function IsMultidimensionalArray(ByVal SourceArray As Variant) As Boolean            ...
    If IsArray(SourceArray) Then
        On Error GoTo ErrHandler
```

```
            Dim LocalUpperBound As Long
            LocalUpperBound = UBound(SourceArray, 2)
            IsMultidimensionalArray = True
            On Error GoTo 0
        End If
        Exit Function
    ErrHandler:
        Err.clear
    End Function
    Private Function Rebase(Optional ByRef SourceArray As Variant, Optional ByVal CurrentArrayType As  _       ...
    ArrayTypes) As Variant()
        Dim LocalItems() As Variant
        Dim LocalType   As ArrayTypes
        If IsMissing(SourceArray) Or Not IsArray(SourceArray) Then
            LocalItems = InternalItems
            LocalType = This.ArrayType
        Else
            LocalItems = SourceArray
            If CurrentArrayType = 0 Then
                LocalType = GetArrayType(LocalItems)
            Else
                LocalType = CurrentArrayType
            End If
        End If
        If LocalType = BA_MULTIDIMENSION Then
            Rebase = RecursiveRebase(LocalItems, True)
        Else
            Rebase = RecursiveRebase(LocalItems, False)
        End If
    End Function
    Private Function RecursiveRebase( _                                                                     ...
            ByRef SourceArray() As Variant, _
            ByVal Recurse As Boolean _
            ) As Variant()
        Dim i           As Long
        Dim LocalLowerBound As Long
        Dim LocalUpperBound As Long
        Dim OFFSET      As Long
        Dim NewItems()  As Variant
        LocalLowerBound = LBound(SourceArray)
        LocalUpperBound = UBound(SourceArray)
        OFFSET = This.LowerBound - LocalLowerBound
        ReDim NewItems(This.LowerBound To LocalUpperBound + OFFSET)
        For i = LocalLowerBound To LocalUpperBound
            If IsArray(SourceArray(i)) And Recurse And OFFSET <> 0 Then
                Dim Nested() As Variant
                Nested = SourceArray(i)
                NewItems(i + OFFSET) = RecursiveRebase(Nested, Recurse)
            Else
                NewItems(i + OFFSET) = SourceArray(i)
            End If
        Next
        RecursiveRebase = NewItems
    End Function
    Private Sub EnsureCapacity(ByVal MinimumCapacity As Long)                                               ...
        If This.Capacity < MinimumCapacity Then
            Dim NewCapacity As Long
            NewCapacity = IIf(This.Capacity = 0, DEFAULT_CAPACITY, This.Capacity * 2)
            If NewCapacity > MAX_ARRAY_LENGTH Then
                NewCapacity = MAX_ARRAY_LENGTH
            End If
            If NewCapacity < MinimumCapacity Then
                NewCapacity = MinimumCapacity
            End If
            Me.Capacity = NewCapacity
        End If
    End Sub
    Private Function ConvertArrayForStorage( _                                                              ...
            ByRef Values As Variant, _
            ByVal CurrentArrayType As ArrayTypes, _
            Optional ByVal ConvertMultiToJagged As Boolean, _
            Optional ByVal ConvertJaggedNestedArrays As Boolean _
            ) As Variant()
        Dim Result()    As Variant
        Select Case CurrentArrayType
            Case ArrayTypes.BA_MULTIDIMENSION
                If ConvertMultiToJagged Then
                    Result = MultiToJagged(Values)
                ElseIf TypeName(Values) <> "Variant()" Then
                    Result = TypedMultiToVariantMulti(Values)
                Else
                    Result = Values
                End If
            Case ArrayTypes.BA_ONEDIMENSION, ArrayTypes.BA_JAGGED
                If TypeName(Values) <> "Variant()" _
                        Or (CurrentArrayType = BA_JAGGED And ConvertJaggedNestedArrays) _
                        Or LBound(Values) <> This.LowerBound Then
                    Result = TypedJaggedToVariantJagged( _
                            Values, _
                            ConvertJaggedNestedArrays, _
                            ConvertMultiToJagged _
                            )
                Else
                    Result = Values
                End If
        End Select
        ConvertArrayForStorage = Result
    End Function
    Private Function TypedJaggedToVariantJagged( _                                           ...
            ByRef SourceArray As Variant, _
            Optional ByVal ConvertJaggedNestedArrays As Boolean, _
            Optional ByVal ConvertMultiToJagged As Boolean _
            ) As Variant()
        Dim Result()    As Variant
        Dim i           As Long
        Dim Bounds(0 To 1) As Long
        Dim OFFSET      As Long
        Bounds(0) = LBound(SourceArray)
        Bounds(1) = UBound(SourceArray)
        OFFSET = This.LowerBound - Bounds(0)
        ReDim Result(Bounds(0) + OFFSET To Bounds(1) + OFFSET)
        For i = Bounds(0) To Bounds(1)
            If IsArray(SourceArray(i)) Then
                Result(i + OFFSET) = ConvertArrayForStorage( _
                        SourceArray(i), _
                        GetArrayType(SourceArray(i)), _
                        ConvertMultiToJagged, _
                        ConvertJaggedNestedArrays _
                        )
            Else
                LetOrSetElement Result(i + OFFSET), SourceArray(i)
            End If
        Next
        TypedJaggedToVariantJagged = Result
    End Function
    Private Function TypedMultiToVariantMulti( _                                             ...
            ByRef SourceArray As Variant _
            ) As Variant()
        Dim LocalDepth  As Long
        Dim Result()    As Variant
        LocalDepth = GetMultidimensionalArrayDepth(SourceArray)
        If LocalDepth = 2 Then
            Dim FirstDimBounds(0 To 1) As Long
            Dim SecondDimBounds(0 To 1) As Long
            Dim FirstDimOffset As Long
            Dim SecondDimOffset As Long
            Dim i       As Long
            Dim j       As Long
            FirstDimBounds(0) = LBound(SourceArray, 1)
            FirstDimBounds(1) = UBound(SourceArray, 1)
            SecondDimBounds(0) = LBound(SourceArray, 2)
            SecondDimBounds(1) = UBound(SourceArray, 2)
            FirstDimOffset = This.LowerBound - FirstDimBounds(0)
            SecondDimOffset = This.LowerBound - SecondDimBounds(0)
            ReDim Result( _
                    FirstDimBounds(0) + FirstDimOffset To FirstDimBounds(1) + FirstDimOffset, _
                    SecondDimBounds(0) + SecondDimOffset To SecondDimBounds(1) + SecondDimOffset _
                    )
            For i = FirstDimBounds(0) To FirstDimBounds(1)
                For j = SecondDimBounds(0) To SecondDimBounds(1)
                    LetOrSetElement Result(i + FirstDimOffset, j + SecondDimOffset), SourceArray(i, j)
                Next
            Next
        ElseIf LocalDepth > 2 And LocalDepth <= 20 Then
            Result = RecursiveTypedMultiToVariantMulti(SourceArray, LocalDepth)
        Else
            RaiseError EC_MAX_DIMENSIONS_LIMIT, "MultiToJagged()", "sourceArray()"
        End If
        TypedMultiToVariantMulti = Result
    End Function
    Private Function MapMultidimensionArray( _                                                  ...
            ByRef SourceArray As Variant, _
            Optional ByVal KnownDepth As Long _
            ) As Variant()
        Dim i           As Long
        Dim LocalMap()  As Variant
        Dim LocalDepth  As Long
        Dim CurrentBounds(0 To 1) As Long
        If Not IsArray(SourceArray) Then Exit Function
        If KnownDepth > 0 Then
            LocalDepth = KnownDepth
        Else
            LocalDepth = GetMultidimensionalArrayDepth(SourceArray)
        End If
        ReDim LocalMap(0 To LocalDepth - 1)
        For i = LBound(LocalMap) To UBound(LocalMap)
            CurrentBounds(0) = LBound(SourceArray, i + 1)
            CurrentBounds(1) = UBound(SourceArray, i + 1)
            LocalMap(i) = CurrentBounds
        Next
        MapMultidimensionArray = LocalMap
    End Function
    Private Function RecursiveTypedMultiToVariantMulti( _                                       ...
            ByRef SourceArray As Variant, _
            Optional ByVal Depth As Long, _
            Optional ByVal CurrentDepth As Long, _
            Optional ByRef Crumbs As Variant, _
            Optional ByRef Result As Variant, _
            Optional ByVal EnsureScalar As Boolean _
            ) As Variant()
        Dim i           As Long
        Dim LocalLowerBound As Long
        Dim LocalUpperBound As Long
        Dim LocalDepth  As Long
        Dim LocalCurrentDepth As Long
        Dim LocalCrumbs() As Variant
        Dim LocalResult() As Variant
        Dim CurrentElement As Variant
        LocalDepth = Depth
        LocalCurrentDepth = CurrentDepth
        If IsMissing(Crumbs) Then
            ReDim LocalCrumbs(LocalDepth - 1)
        Else
            LocalCrumbs = Crumbs
        End If
        If IsMissing(Result) Then
            LocalResult = CreateMultidimensionalArray(MapMultidimensionArray(SourceArray, LocalDepth))
        Else
            LocalResult = Result
        End If
        inc LocalCurrentDepth
        LocalLowerBound = LBound(SourceArray, LocalCurrentDepth)
        LocalUpperBound = UBound(SourceArray, LocalCurrentDepth)
        For i = LocalLowerBound To LocalUpperBound
            LocalCrumbs(LocalCurrentDepth - 1) = i
            If LocalCurrentDepth = LocalDepth Then
```

```
            LetOrSetElement CurrentElement, GetElementByBreadcrumb(SourceArray, LocalCrumbs)
            If EnsureScalar Then
                LocalResult = LetElementByBreadcrumb(LocalResult, LocalCrumbs,  _
                GetScalarRepresentation(CurrentElement))
            Else
                LocalResult = LetElementByBreadcrumb(LocalResult, LocalCrumbs, CurrentElement)
            End If
        Else
            LocalResult = RecursiveTypedMultiToVariantMulti( _
                    SourceArray:=SourceArray, _
                    Depth:=LocalDepth, _
                    CurrentDepth:=LocalCurrentDepth, _
                    Crumbs:=LocalCrumbs, _
                    Result:=LocalResult, _
                    EnsureScalar:=EnsureScalar _
                    )
        End If
    Next
    RecursiveTypedMultiToVariantMulti = LocalResult
End Function
Private Function JaggedToMulti( _
        ByRef SourceArray() As Variant, _
        Optional ByVal Depth As Long, _
        Optional ByVal EnsureScalar As Boolean _
        ) As Variant()
    Dim LocalDepth  As Long
    Dim Result()    As Variant
    LocalDepth = Depth
    If LocalDepth = 0 Then
        If Not IsJaggedArray(SourceArray) Then
            RaiseError EC_EXPECTED_JAGGED_ARRAY, "JaggedToMulti", "sourceArray()"
            Exit Function
        End If
        LocalDepth = GetJaggedArrayDepth(SourceArray)
    End If
    If LocalDepth <= 1 Then
        Result = SourceArray
    ElseIf LocalDepth = 2 Then
        Dim FirstDimBounds() As Long
        Dim SecondDimBounds() As Long
        Dim i       As Long
        Dim j       As Long
        FirstDimBounds = GetArrayBounds(SourceArray)
        SecondDimBounds = GetMaxBoundsAtDimension(SourceArray, 2)
        ReDim Result(FirstDimBounds(0) To FirstDimBounds(1), _
                SecondDimBounds(0) To SecondDimBounds(1))
        For i = FirstDimBounds(0) To FirstDimBounds(1)
            If IsArray(SourceArray(i)) Then
                For j = LBound(SourceArray(i)) To UBound(SourceArray(i))
                    If EnsureScalar Then
                        LetOrSetElement Result(i, j), _
                                GetScalarRepresentation(SourceArray(i)(j))
                    Else
                        LetOrSetElement Result(i, j), _
                                SourceArray(i)(j)
                    End If
                Next
            Else
                If EnsureScalar Then
                    LetOrSetElement Result(i, LBound(Result, 2)), _
                            GetScalarRepresentation(SourceArray(i))
                Else
                    LetOrSetElement Result(i, LBound(Result, 2)), _
                            SourceArray(i)
                End If
            End If
        Next
    ElseIf LocalDepth > 2 And LocalDepth <= 20 Then
        Result = RecursiveJaggedToMulti(SourceArray, LocalDepth, EnsureScalar:=EnsureScalar)
    Else
        RaiseError EC_MAX_DIMENSIONS_LIMIT, "JaggedToMulti()", "sourceArray()"
        Result = SourceArray
    End If
    JaggedToMulti = Result
End Function
Private Function MultiToJagged( _                                                                   ...
        ByRef SourceArray As Variant, _
        Optional ByVal Depth As Long _
        ) As Variant()
    Dim LocalDepth  As Long
    Dim Result()    As Variant
    LocalDepth = Depth
    If LocalDepth = 0 Then
        If Not IsMultidimensionalArray(SourceArray) Then
            RaiseError EC_EXPECTED_MULTIDIMENSION_ARRAY, "MultiToJagged", "sourceArray()"
            Result = SourceArray
            Exit Function
        End If
        LocalDepth = GetMultidimensionalArrayDepth(SourceArray)
    End If
    If LocalDepth <= 1 Then
        Result = SourceArray
    ElseIf LocalDepth = 2 Then
        Dim FirstDimBounds(0 To 1) As Long
        Dim SecondDimBounds(0 To 1) As Long
        Dim FirstDimOffset As Long
        Dim SecondDimOffset As Long
        Dim Nested() As Variant
        Dim i       As Long
        Dim j       As Long
        FirstDimBounds(0) = LBound(SourceArray, 1)
        FirstDimBounds(1) = UBound(SourceArray, 1)
        SecondDimBounds(0) = LBound(SourceArray, 2)
        SecondDimBounds(1) = UBound(SourceArray, 2)
        FirstDimOffset = This.LowerBound - FirstDimBounds(0)
        SecondDimOffset = This.LowerBound - SecondDimBounds(0)
        ReDim Result(FirstDimBounds(0) + FirstDimOffset To FirstDimBounds(1) + FirstDimOffset)
        For i = FirstDimBounds(0) To FirstDimBounds(1)
            ReDim Nested(SecondDimBounds(0) + SecondDimOffset To SecondDimBounds(1) +  _
            SecondDimOffset)
            For j = SecondDimBounds(0) To SecondDimBounds(1)
                LetOrSetElement Nested(j + SecondDimOffset), SourceArray(i, j)
            Next
            Result(i + FirstDimOffset) = Nested
        Next
    ElseIf LocalDepth > 2 And LocalDepth <= 20 Then
        Result = RecursiveMultiToJagged(SourceArray, LocalDepth)
    Else
        RaiseError EC_MAX_DIMENSIONS_LIMIT, "MultiToJagged()", "sourceArray()"
        Result = SourceArray
    End If
    MultiToJagged = Result
End Function
Private Function RecursiveMultiToJagged( _                                                          ...
        ByRef SourceArray As Variant, _
        Optional ByVal Depth As Long, _
        Optional ByVal CurrentDepth As Long, _
        Optional ByRef Crumbs As Variant _
        ) As Variant()
    Dim i           As Long
    Dim LocalLowerBound As Long
    Dim LocalUpperBound As Long
    Dim LocalDepth  As Long
    Dim LocalCurrentDepth As Long
    Dim LocalCrumbs() As Variant
    Dim Result()    As Variant
    Dim BoundOffset As Long
    LocalDepth = Depth
    LocalCurrentDepth = CurrentDepth
    If LocalDepth > 1 Then
        If IsMissing(Crumbs) Then
            ReDim LocalCrumbs(LocalDepth - 1)
        Else
            LocalCrumbs = Crumbs
        End If
        inc LocalCurrentDepth
        LocalLowerBound = LBound(SourceArray, LocalCurrentDepth)
        LocalUpperBound = UBound(SourceArray, LocalCurrentDepth)
        BoundOffset = This.LowerBound - LocalLowerBound
        ReDim Result(LocalLowerBound + BoundOffset To LocalUpperBound + BoundOffset)
        For i = LocalLowerBound To LocalUpperBound
            LocalCrumbs(LocalCurrentDepth - 1) = i
            If LocalCurrentDepth = LocalDepth Then
                LetOrSetElement Result(i + BoundOffset), GetElementByBreadcrumb(SourceArray,  _
                LocalCrumbs)
            Else
                Result(i + BoundOffset) = RecursiveMultiToJagged(SourceArray, LocalDepth,  _
                LocalCurrentDepth, LocalCrumbs)
            End If
        Next
    Else
        Result = SourceArray
    End If
    RecursiveMultiToJagged = Result
End Function
Private Function RecursiveJaggedToMulti( _                                                          …
        ByRef SourceArray() As Variant, _
        Optional ByVal Depth As Long, _
        Optional ByVal CurrentDepth As Long, _
        Optional ByRef Crumbs As Variant, _
        Optional ByRef Result As Variant, _
        Optional ByVal EnsureScalar As Boolean _
        ) As Variant()
    Dim i           As Long
    Dim LocalLowerBound As Long
    Dim LocalUpperBound As Long
    Dim LocalDepth  As Long
    Dim LocalCurrentDepth As Long
    Dim LocalCrumbs() As Variant
    Dim LocalResult() As Variant
    LocalDepth = Depth
    LocalCurrentDepth = CurrentDepth
    If IsMissing(Crumbs) Then
        ReDim LocalCrumbs(LocalDepth - 1)
    Else
        LocalCrumbs = Crumbs
    End If
    If IsMissing(Result) Then
        LocalResult = CreateMultidimensionalArray(MapJaggedArray(SourceArray, KnownDepth:= _
        LocalDepth))
    Else
        LocalResult = Result
    End If
    inc LocalCurrentDepth
    LocalLowerBound = LBound(SourceArray)
    LocalUpperBound = UBound(SourceArray)
    For i = LocalLowerBound To LocalUpperBound
        LocalCrumbs(LocalCurrentDepth - 1) = i
        If LocalCurrentDepth = LocalDepth Then
            If EnsureScalar Then
                LocalResult = LetElementByBreadcrumb(LocalResult, LocalCrumbs,  _
                GetScalarRepresentation(SourceArray(i)))
            Else
                LocalResult = LetElementByBreadcrumb(LocalResult, LocalCrumbs, SourceArray(i))
            End If
        Else
            Dim Nested() As Variant
            If Not IsArray(SourceArray(i)) Then
                Nested = Array(SourceArray(i))
            Else
                Nested = SourceArray(i)
            End If
            LocalResult = RecursiveJaggedToMulti( _
                    SourceArray:=Nested, _
                    Depth:=LocalDepth, _
                    CurrentDepth:=LocalCurrentDepth, _
                    Crumbs:=LocalCrumbs, _
                    Result:=LocalResult, _
                    EnsureScalar:=EnsureScalar _
                    )
        End If
    Next
    RecursiveJaggedToMulti = LocalResult
End Function
Private Function GetJaggedArrayDepth(ByRef SourceArray() As Variant) As Long                                        ...
    Dim i            As Long
    Dim LocalLowerBound As Long
    Dim LocalUpperBound As Long
    Dim CurrentDepth As Long
    Dim MaxDepth     As Long
    Dim Depth        As Long
    If IsArray(SourceArray) Then
        inc Depth
        LocalLowerBound = LBound(SourceArray)
        LocalUpperBound = UBound(SourceArray)
        For i = LocalLowerBound To LocalUpperBound
            If IsArray(SourceArray(i)) Then
                Dim Nested() As Variant
                Nested = SourceArray(i)
                CurrentDepth = GetJaggedArrayDepth(Nested)
                If CurrentDepth > MaxDepth Then MaxDepth = CurrentDepth
            End If
        Next
        inc Depth, MaxDepth
    End If
    GetJaggedArrayDepth = Depth
End Function
Private Function GetMaxBoundsAtDimension( _                                                                          ...
        ByRef SourceArray() As Variant, _
        Optional ByVal Dimension As Long = 1, _
        Optional ByVal CurrentDimension As Long = 1 _
        ) As Long()
    Dim MaxBounds() As Long
    If CurrentDimension < Dimension Then
        Dim i        As Long
        Dim CurrentBounds() As Long
        Dim LocalLowerBound As Long
        Dim LocalUpperBound As Long
        LocalLowerBound = LBound(SourceArray)
        LocalUpperBound = UBound(SourceArray)
        For i = LocalLowerBound To LocalUpperBound
            If IsArray(SourceArray(i)) Then
                Dim Nested() As Variant
                Nested = SourceArray(i)
                CurrentBounds = GetMaxBoundsAtDimension(Nested, Dimension, CurrentDimension + 1)
                If Not IsArrayAllocated(MaxBounds) Then
                    MaxBounds = CurrentBounds
                Else
                    If CurrentBounds(0) < MaxBounds(0) Then MaxBounds(0) = CurrentBounds(0)
                    If CurrentBounds(1) > MaxBounds(1) Then MaxBounds(1) = CurrentBounds(1)
                End If
            End If
        Next
    Else
        MaxBounds = GetArrayBounds(SourceArray)
    End If
    GetMaxBoundsAtDimension = MaxBounds
End Function
Private Function MapJaggedArray( _                                                                                   ...
        ByRef SourceArray() As Variant, _
        Optional ByVal KnownDepth As Long _
        ) As Variant()
    Dim i            As Long
    Dim LocalMap()   As Variant
    Dim LocalDepth   As Long
    If Not IsArray(SourceArray) Then Exit Function
    If KnownDepth > 0 Then
        LocalDepth = KnownDepth
    Else
        LocalDepth = GetJaggedArrayDepth(SourceArray)
    End If
    ReDim LocalMap(0 To LocalDepth - 1)
    For i = LBound(LocalMap) To UBound(LocalMap)
        LocalMap(i) = GetMaxBoundsAtDimension(SourceArray, i + 1)
    Next
    MapJaggedArray = LocalMap
End Function
Private Function GetArrayBounds(ByVal SourceArray As Variant) As Long()                                     ...
    Dim Result(0 To 1) As Long
    If IsArray(SourceArray) Then
        Result(0) = LBound(SourceArray)
        Result(1) = UBound(SourceArray)
    End If
    GetArrayBounds = Result
End Function
Private Function GetMultidimensionalArrayDepth(ByVal SourceArray As Variant) As Long                        ...
    Dim i           As Long
    Dim Void        As Long
    On Error Resume Next
    Do
        inc i
        Void = UBound(SourceArray, i)
    Loop Until Err.Number <> 0
    Err.clear
    On Error GoTo 0
    GetMultidimensionalArrayDepth = i - 1
End Function
Private Function GetElementByBreadcrumb( _                                                                  ...
        ByVal SourceArray As Variant, _
        ByRef Crumb() As Variant _
        ) As Variant
    Dim Result      As Variant
    Select Case UBound(Crumb)
        Case 0
            LetOrSetElement Result, SourceArray(Crumb(0))
        Case 1
            LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1))
        Case 2
            LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2))
        Case 3
            LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3))
        Case 4
            LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4))
        Case 5
            LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
            Crumb(5))
        Case 6
            LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
            Crumb(5), Crumb(6))
        Case 7
            LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
            Crumb(5), Crumb(6), Crumb(7))
        Case 8
            LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
            Crumb(5), Crumb(6), Crumb(7), Crumb(8))
        Case 9
            LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
            Crumb(5), Crumb(6), Crumb(7), Crumb(8), _
                    Crumb(9))
        Case 10
            LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
            Crumb(5), Crumb(6), Crumb(7), Crumb(8), _
                    Crumb(9), Crumb(10))
        Case 11
            LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
            Crumb(5), Crumb(6), Crumb(7), Crumb(8), _
                    Crumb(9), Crumb(10), Crumb(11))
        Case 12
            LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
            Crumb(5), Crumb(6), Crumb(7), Crumb(8), _
```

```
                    Crumb(9), Crumb(10), Crumb(11), Crumb(12))
            Case 13
                LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
                Crumb(5), Crumb(6), Crumb(7), Crumb(8), _
                        Crumb(9), Crumb(10), Crumb(11), Crumb(12), Crumb(13))
            Case 14
                LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
                Crumb(5), Crumb(6), Crumb(7), Crumb(8), _
                        Crumb(9), Crumb(10), Crumb(11), Crumb(12), Crumb(13), Crumb(14))
            Case 15
                LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
                Crumb(5), Crumb(6), Crumb(7), Crumb(8), _
                        Crumb(9), Crumb(10), Crumb(11), Crumb(12), Crumb(13), Crumb(14), Crumb(15))
            Case 16
                LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
                Crumb(5), Crumb(6), Crumb(7), Crumb(8), _
                        Crumb(9), Crumb(10), Crumb(11), Crumb(12), Crumb(13), Crumb(14), Crumb(15),  _
                        Crumb(16))
            Case 17
                LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
                Crumb(5), Crumb(6), Crumb(7), Crumb(8), _
                        Crumb(9), Crumb(10), Crumb(11), Crumb(12), Crumb(13), Crumb(14), Crumb(15),  _
                        Crumb(16), Crumb(17))
            Case 18
                LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
                Crumb(5), Crumb(6), Crumb(7), Crumb(8), _
                        Crumb(9), Crumb(10), Crumb(11), Crumb(12), Crumb(13), Crumb(14), Crumb(15),  _
                        Crumb(16), Crumb(17), Crumb(18))
            Case 19
                LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
                Crumb(5), Crumb(6), Crumb(7), Crumb(8), _
                        Crumb(9), Crumb(10), Crumb(11), Crumb(12), Crumb(13), Crumb(14), Crumb(15),  _
                        Crumb(16), Crumb(17), Crumb(18), Crumb(19))
            Case 20
                LetOrSetElement Result, SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4),  _
                Crumb(5), Crumb(6), Crumb(7), Crumb(8), _
                        Crumb(9), Crumb(10), Crumb(11), Crumb(12), Crumb(13), Crumb(14), Crumb(15),  _
                        Crumb(16), Crumb(17), Crumb(18), Crumb(19), Crumb(20))
        End Select
        GetElementByBreadcrumb = Result
    End Function
    Private Function LetElementByBreadcrumb( _                                                  ...
            ByRef SourceArray() As Variant, _
            ByRef Crumb() As Variant, _
            ByVal Element As Variant _
            ) As Variant
        Select Case UBound(Crumb)
            Case 0
                LetOrSetElement SourceArray(Crumb(0)), Element
            Case 1
                LetOrSetElement SourceArray(Crumb(0), Crumb(1)), Element
            Case 2
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2)), Element
            Case 3
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3)), Element
            Case 4
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4)), Element
            Case 5
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5)), _
                Element
            Case 6
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5),  _
                Crumb(6)), Element
            Case 7
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5),  _
                Crumb(6), Crumb(7)), Element
            Case 8
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5),  _
                Crumb(6), Crumb(7), Crumb(8)), Element
            Case 9
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5),  _
                Crumb(6), Crumb(7), Crumb(8), Crumb(9)), _
                        Element
            Case 10
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5),  _
                Crumb(6), Crumb(7), Crumb(8), Crumb(9), _
                        Crumb(10)), Element
            Case 11
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5),  _
                Crumb(6), Crumb(7), Crumb(8), Crumb(9), _
                        Crumb(10), Crumb(11)), Element
            Case 12
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5),  _
                Crumb(6), Crumb(7), Crumb(8), Crumb(9), _
                        Crumb(10), Crumb(11), Crumb(12)), Element
            Case 13
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5),  _
                Crumb(6), Crumb(7), Crumb(8), Crumb(9), _
                        Crumb(10), Crumb(11), Crumb(12), Crumb(13)), Element
            Case 14
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5),  _
                Crumb(6), Crumb(7), Crumb(8), Crumb(9), _
                        Crumb(10), Crumb(11), Crumb(12), Crumb(13), Crumb(14)), Element
            Case 15
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5),  _
                Crumb(6), Crumb(7), Crumb(8), Crumb(9), _
                        Crumb(10), Crumb(11), Crumb(12), Crumb(13), Crumb(14), Crumb(15)), Element
            Case 16
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5),  _
                Crumb(6), Crumb(7), Crumb(8), Crumb(9), _
                        Crumb(10), Crumb(11), Crumb(12), Crumb(13), Crumb(14), Crumb(15), Crumb(16)),  _
                        Element
            Case 17
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5),  _
                Crumb(6), Crumb(7), Crumb(8), Crumb(9), _
                        Crumb(10), Crumb(11), Crumb(12), Crumb(13), Crumb(14), Crumb(15), Crumb(16),  _
                        Crumb(17)), Element
            Case 18
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5),  _
                Crumb(6), Crumb(7), Crumb(8), Crumb(9), _
                        Crumb(10), Crumb(11), Crumb(12), Crumb(13), Crumb(14), Crumb(15), Crumb(16),  _
                        Crumb(17), Crumb(18)), Element
            Case 19
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5),  _
                Crumb(6), Crumb(7), Crumb(8), Crumb(9), _
                        Crumb(10), Crumb(11), Crumb(12), Crumb(13), Crumb(14), Crumb(15), Crumb(16),  _
                        Crumb(17), Crumb(18), Crumb(19)), Element
            Case 20
                LetOrSetElement SourceArray(Crumb(0), Crumb(1), Crumb(2), Crumb(3), Crumb(4), Crumb(5),  _
                Crumb(6), Crumb(7), Crumb(8), Crumb(9), _
                        Crumb(10), Crumb(11), Crumb(12), Crumb(13), Crumb(14), Crumb(15), Crumb(16),  _
                        Crumb(17), Crumb(18), Crumb(19), Crumb(20)), _
                        Element
        End Select
        LetElementByBreadcrumb = SourceArray
    End Function
    Private Function CreateMultidimensionalArray(ByRef Crumb() As Variant) As Variant()               …
        Dim Result()    As Variant
        Select Case UBound(Crumb)
            Case 0
                ReDim Result(Crumb(0)(0) To Crumb(0)(1))
            Case 1
                ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1))
            Case 2
                ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
                Crumb(2)(1))
            Case 3
                ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
                Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1))
            Case 4
                ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
                Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                        Crumb(4)(0) To Crumb(4)(1))
            Case 5
```

```
                ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
                Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                        Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1))
            Case 6
                ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
                Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                        Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1), Crumb(6)(0) To Crumb(6)( _
                        1))
            Case 7
                ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
                Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                        Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1), Crumb(6)(0) To Crumb(6)( _
                        1), Crumb(7)(0) To Crumb(7)(1))
            Case 8
                ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
                Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                        Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1), Crumb(6)(0) To Crumb(6)( _
                        1), Crumb(7)(0) To Crumb(7)(1), _
                        Crumb(8)(0) To Crumb(8)(1))
            Case 9
                ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
                Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                        Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1), Crumb(6)(0) To Crumb(6)( _
                        1), Crumb(7)(0) To Crumb(7)(1), _
                        Crumb(8)(0) To Crumb(8)(1), Crumb(9)(0) To Crumb(9)(1))
            Case 10
                ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
                Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                        Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1), Crumb(6)(0) To Crumb(6)( _
                        1), Crumb(7)(0) To Crumb(7)(1), _
                        Crumb(8)(0) To Crumb(8)(1), Crumb(9)(0) To Crumb(9)(1), Crumb(10)(0) To Crumb( _
                        10)(1))
            Case 11
                ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
                Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                        Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1), Crumb(6)(0) To Crumb(6)( _
                        1), Crumb(7)(0) To Crumb(7)(1), _
                        Crumb(8)(0) To Crumb(8)(1), Crumb(9)(0) To Crumb(9)(1), Crumb(10)(0) To Crumb( _
                        10)(1), Crumb(11)(0) To Crumb(11)(1))
            Case 12
                ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
                Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                        Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1), Crumb(6)(0) To Crumb(6)( _
                        1), Crumb(7)(0) To Crumb(7)(1), _
                        Crumb(8)(0) To Crumb(8)(1), Crumb(9)(0) To Crumb(9)(1), Crumb(10)(0) To Crumb( _
                        10)(1), Crumb(11)(0) To Crumb(11)(1), _
                        Crumb(12)(0) To Crumb(12)(1))
            Case 13
                ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
                Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                        Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1), Crumb(6)(0) To Crumb(6)( _
                        1), Crumb(7)(0) To Crumb(7)(1), _
                        Crumb(8)(0) To Crumb(8)(1), Crumb(9)(0) To Crumb(9)(1), Crumb(10)(0) To Crumb( _
                        10)(1), Crumb(11)(0) To Crumb(11)(1), _
                        Crumb(12)(0) To Crumb(12)(1), Crumb(13)(0) To Crumb(13)(1))
            Case 14
                ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
                Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                        Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1), Crumb(6)(0) To Crumb(6)( _
                        1), Crumb(7)(0) To Crumb(7)(1), _
                        Crumb(8)(0) To Crumb(8)(1), Crumb(9)(0) To Crumb(9)(1), Crumb(10)(0) To Crumb( _
                        10)(1), Crumb(11)(0) To Crumb(11)(1), _
                        Crumb(12)(0) To Crumb(12)(1), Crumb(13)(0) To Crumb(13)(1), Crumb(14)(0) To  _
                        Crumb(14)(1))
            Case 15
                ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
                Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                        Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1), Crumb(6)(0) To Crumb(6)( _
                        1), Crumb(7)(0) To Crumb(7)(1), _
                        Crumb(8)(0) To Crumb(8)(1), Crumb(9)(0) To Crumb(9)(1), Crumb(10)(0) To Crumb( _
                        10)(1), Crumb(11)(0) To Crumb(11)(1), _
                        Crumb(12)(0) To Crumb(12)(1), Crumb(13)(0) To Crumb(13)(1), Crumb(14)(0) To  _
```

```
                    Crumb(14)(1), Crumb(15)(0) To Crumb(15)(1))
        Case 16
            ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
            Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                    Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1), Crumb(6)(0) To Crumb(6)( _
                    1), Crumb(7)(0) To Crumb(7)(1), _
                    Crumb(8)(0) To Crumb(8)(1), Crumb(9)(0) To Crumb(9)(1), Crumb(10)(0) To Crumb( _
                    10)(1), Crumb(11)(0) To Crumb(11)(1), _
                    Crumb(12)(0) To Crumb(12)(1), Crumb(13)(0) To Crumb(13)(1), Crumb(14)(0) To  _
                    Crumb(14)(1), Crumb(15)(0) To Crumb(15)(1), _
                    Crumb(16)(0) To Crumb(16)(1))
        Case 17
            ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
            Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                    Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1), Crumb(6)(0) To Crumb(6)( _
                    1), Crumb(7)(0) To Crumb(7)(1), _
                    Crumb(8)(0) To Crumb(8)(1), Crumb(9)(0) To Crumb(9)(1), Crumb(10)(0) To Crumb( _
                    10)(1), Crumb(11)(0) To Crumb(11)(1), _
                    Crumb(12)(0) To Crumb(12)(1), Crumb(13)(0) To Crumb(13)(1), Crumb(14)(0) To  _
                    Crumb(14)(1), Crumb(15)(0) To Crumb(15)(1), _
                    Crumb(16)(0) To Crumb(16)(1), Crumb(17)(0) To Crumb(17)(1))
        Case 18
            ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
            Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                    Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1), Crumb(6)(0) To Crumb(6)( _
                    1), Crumb(7)(0) To Crumb(7)(1), _
                    Crumb(8)(0) To Crumb(8)(1), Crumb(9)(0) To Crumb(9)(1), Crumb(10)(0) To Crumb( _
                    10)(1), Crumb(11)(0) To Crumb(11)(1), _
                    Crumb(12)(0) To Crumb(12)(1), Crumb(13)(0) To Crumb(13)(1), Crumb(14)(0) To  _
                    Crumb(14)(1), Crumb(15)(0) To Crumb(15)(1), _
                    Crumb(16)(0) To Crumb(16)(1), Crumb(17)(0) To Crumb(17)(1), Crumb(18)(0) To  _
                    Crumb(18)(1))
        Case 19
            ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
            Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                    Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1), Crumb(6)(0) To Crumb(6)( _
                    1), Crumb(7)(0) To Crumb(7)(1), _
                    Crumb(8)(0) To Crumb(8)(1), Crumb(9)(0) To Crumb(9)(1), Crumb(10)(0) To Crumb( _
                    10)(1), Crumb(11)(0) To Crumb(11)(1), _
                    Crumb(12)(0) To Crumb(12)(1), Crumb(13)(0) To Crumb(13)(1), Crumb(14)(0) To  _
                    Crumb(14)(1), Crumb(15)(0) To Crumb(15)(1), _
                    Crumb(16)(0) To Crumb(16)(1), Crumb(17)(0) To Crumb(17)(1), Crumb(18)(0) To  _
                    Crumb(18)(1), Crumb(19)(0) To Crumb(19)(1))
        Case 20
            ReDim Result(Crumb(0)(0) To Crumb(0)(1), Crumb(1)(0) To Crumb(1)(1), Crumb(2)(0) To  _
            Crumb(2)(1), Crumb(3)(0) To Crumb(3)(1), _
                    Crumb(4)(0) To Crumb(4)(1), Crumb(5)(0) To Crumb(5)(1), Crumb(6)(0) To Crumb(6)( _
                    1), Crumb(7)(0) To Crumb(7)(1), _
                    Crumb(8)(0) To Crumb(8)(1), Crumb(9)(0) To Crumb(9)(1), Crumb(10)(0) To Crumb( _
                    10)(1), Crumb(11)(0) To Crumb(11)(1), _
                    Crumb(12)(0) To Crumb(12)(1), Crumb(13)(0) To Crumb(13)(1), Crumb(14)(0) To  _
                    Crumb(14)(1), Crumb(15)(0) To Crumb(15)(1), _
                    Crumb(16)(0) To Crumb(16)(1), Crumb(17)(0) To Crumb(17)(1), Crumb(18)(0) To  _
                    Crumb(18)(1), Crumb(19)(0) To Crumb(19)(1), _
                    Crumb(20)(0) To Crumb(20)(1))
    End Select
    CreateMultidimensionalArray = Result
End Function
```