Pearson Software Consulting Services

Functions For VBA Arrays

If you're writing anything but the most trivial VBA procedures, it is quite likely that you will be using arrays in your VBA code to store data or series of related data. This page describes nearly 40 functions you can use to get information about and manipulate arrays. It is assumed that you know the basics of VBA arrays. For information about passing and returning arrays to and from procedures, see the Passing And Returning Arrays With Functions page.

The following terminology used on this page:

A ***static*** array is an array that is sized in the `Dim` statement that declares the array. E.g.,
`Dim StaticArray(1 To 10) As Long`
You cannot change the size or data type of a static array. When you Erase a static array, no memory is freed. Erase simple set all the elements to their default value (0, vbNullString, Empty, or Nothing, depending on the data type of the array).

A ***dynamic*** array is an array that is not sized in the `Dim` statement. Instead, it is sized with the `ReDim` statement. E.g.,
`Dim DynamicArray() As Long`
`ReDim DynamicArray(1 To 10)`
You can change the size of a dynamic array, but not the data type. When you Erase a dynamic array, the memory allocated to the array is released. You must ReDim the array in order to use it after it has been Erased.

An array is ***allocated*** if it is either a static array or a dynamic array that has been sized with the `ReDim` statement.
Static arrays are always allocated and never empty.

An array is ***empty*** or ***unallocated*** if it is a dynamic array that has not yet been sized with the `ReDim` statement or that has been deallocated with the Erase statement. Static arrays are never unallocated or empty.

An ***element*** is one specific item in an array of items.

This page describes about 30 functions that you can use to get information about and manipulate arrays. You can download a bas module file containing the procedures here. The downloadable file contains two modules: modArraySupport, which contains all of the VBA code, and modDemo which contains procedures testing and illustrating the function in modArraySupport. You do not need to include modDemo in your project in order to use modArraySupport.

These functions call upon one another, so it is recommended that you Import the entire module file into your project.

This page describes the following procedures:

```
AreDataTypesCompatible
ChangeBoundsOfArray
CombineTwoDArrays
CompareArrays
ConcatenateArrays
CopyArray
CopyArraySubSetToArray
CopyNonNothingObjectsToArray
DataTypeOfArray
DeleteArrayElement
ExpandArray
FirstNonEmptyStringIndexInArray
GetColumn
GetRow
InsertElementIntoArray
IsArrayAllDefault
IsArrayAllNumeric
IsArrayAllocated
IsArrayDynamic
IsArrayEmpty
IsArrayObjects
IsArraySorted
IsNumericDataType
IsVariantArrayConsistent
IsVariantArrayNumeric
MoveEmptyStringsToEndOfArray
NumberOfArrayDimensions
NumElements
ResetVariantArrayToDefaults
ReverseArrayInPlace
ReverseArrayOfObjectsInPlace
SetObjectArrayToNothing
SetVariableToDefault
SwapArrayRows
SwapArrayColumns
TransposeArray
VectorsToArray
```

If you are new VBA (or VB) arrays, but have experience with arrays in other programming languages (e.g., C), you will find that VBA arrays work pretty much the same. The primary difference is that a VBA array is more than just a sequential series of bytes. A VBA array is actually a structure called a SAFEARRAY which contains information about the array including the number of dimensions and the number of elements in each dimension. This structure includes a pointer variable that points to the to the actual data. Any array operation in your VBA code uses the appropriate SAFEARRAY API functions. These are documented on MSDN. While this may add some overhead processing to the project, it prevents common bugs that are frequent in standard arrays, such as going beyond the end of the array. An attempt to access an element beyond the end of the array will result in a trappable run-time error "9 -- Subscript out of range".

Another significant difference between VB/VBA arrays and conventional (e.g., C) arrays, is that you can specify any value for the lower and upper bounds of the array. Element 0 need not be the first element in the array. For example, the following is perfectly legal code (as long as the lower bound is less than or equal to the upper bound -- you'll receive a compiler error if the lower bound is greater the upper bound):

```
Dim N As Long
Dim Arr(-100 To -51) As Long
Debug.Print "LBound: " & CStr(LBound(Arr)), _
    "UBound: " & CStr(UBound(Arr)), _
    "NumElements: " & CStr(UBound(Arr) - LBound(Arr) + 1)
For N = LBound(Arr) To UBound(Arr)
    Arr(N) = N * 100
Next N
```

While I have never found the need to use a lower bound other than 0 or 1, there are circumstances in which this might be useful, and VB/VBA array will support it.

The final significant difference is that if you don't explicitly declare the lower bound of an array, the lower bound will be assumed to be either 0 or 1, depending on value of the Option Base statement, if present. If Option Base is not present in the module, 0 is assumed. For example, the code

```
Dim Arr(10) As Long
```

declares an array of either 10 or 11 elements. Note that the declaration does ***not*** specify the number of elements in the array. Instead, it specifies the upper bound of the array. If your module does not contain an Option Base statement, the lower bound is assumed to be zero, and the declaration above is the same as

```
Dim Arr(0 To 10) As Long
```

If you have an Option Base statement of 0 or 1, the lower bound of the array is set to that value. Thus, the code

```
Dim Arr(10) As Long
```

is is the equivalent of either

```
Dim Arr(0 To 10) As Long
' or
Dim Arr(1 To 10) As Long
```

depending on the value of the Option Base. It is, in my opinion, very poor programming practice to omit the lower bound and declare only the upper bound. Omitting the lower bound will lead to bugs when you copy/paste code between modules and projects. You should always explicitly specify both the lower and upper bound for the array, either in the Dim or a ReDim statement.

Finally, because the lower and upper bounds of a dynamic array may be changed at run-time with the ReDim statement, you should *always* use LBound and UBound when looping through an array. Never hard-code array limits. E.g.,

```
Dim N As Long
Dim Arr(-100 To -51) As Long
For N = LBound(Arr) To UBound(Arr)
    ' do something with Arr(N)
Next N
```

Prior to attempting to loop through a dynamically declared array, you should test to ensure that the array has, in fact, been allocated. You can use the IsArrayAllocated function shown below to test this condition:

```
Dim Arr() As Long
If IsArrayAllocated(Arr:=Arr) = True Then
    ' loop through the array
Else
    ' code for unallocated array
End If
```

Function Descriptions

The function descriptions are as follows:

`AreDataTypesCompatible`

```
Public Function AreDataTypesCompatible(DestVar As Variant, SourceVar As Variant) As Boolean
```

This function examines two variables, DestVar and SourceVar, and determines whether they are compatible. The variables are compatible if both variables are the same data type, or if the value in SourceVar can be stored in DestVar without losing precision or encountering an overflow error. For example, a Source Integer is compatible with a Dest Long because an Integer can be stored in a Long variable without loss of precision or overflow. A Source Double is not compatible with a Dest Long because the Double would lose precision (the fractional part of the number will be lost) and its conversion might cause an overflow error.

`ChangeBoundsOfArray`

```
Public Function ChangeBoundsOfArray(InputArr As Variant, _
    NewLowerBound As Long, NewUpperBound) As Boolean
```

This function changes the upper and lower bounds of InputArray. Existing data in InputArr is preserved. InputArr must be a dynamic, allocated single-dimensional array. If the new size of the array (`NewUpperBound-NewLowerBound+1`) is greater than the original array, the additional elements on the right end of the array are set to the default value of the data type of the array (0, vbNullString, Empty, or Nothing). If the new size of the array is less than the size of the original array, the new array will contain only the left-most values of the original array. Elements to the right are lost. The elements of the array may be simple type variables (e.g., Longs, Strings), Objects, or Arrays. User-Defined Type are not allowed. An error will occur if InputArr is not an array, if InputArr is a static array, if InputArr is not allocated, if NewLowerBound is greater than NewUpperBound, or if InputArr is not single-dimensional. The function returns False if an error occurred, or True if the operation was successful.

`CombineTwoDArrays`

```
Public Function CombineTwoDArrays(Arr1 As Variant, _
    Arr2 As Variant) As Variant
```

This function combines two 2-dimensional arrays into a single array. The function returns a Variant containing an array that is the combination of Arr1 and Arr2. If an error occurs, the result is NULL. Both dimensions of both Arr1 and Arr2 must have the same LBound -- that is, all 4 LBounds must be equal. The result array is Arr1 with addition rows appended from Arr2. For example, the arrays

```
a    b              and              e   f
c    d                               g   h
```

are combined to create an array:

```
a    b
c    d
e    f
g    h
```

You can nest calls to CombineTwoDArrays to concatenate several arrays into a singe array. For example,

```
    V = CombineTwoDArrays(CombineTwoDArrays(CombineTwoDArrays(A, B), C), D)
```

`CompareArrays`

```
Public Function CompareArrays(Array1 As Variant, Array2 As Variant, _
    ResultArray As Variant, Optional CompareMode As VbCompareMethod = vbTextCompare) As Boolean
```

This function compares two array, Array1 and Array2, and populates ResultArray with the comparison results of pair of corresponding elements in Array1 and Array2. Each element in Array1 is compare to the corresponding element in Array2, and the corresponding element in ResultArray is set to -1 if the element in Array1 is less than the element in Array2, 0 if the two elements are equal, and +1 if the element in Array1 is greater than the element in Array2. Array1 and Array2 have the same LBound and have the same number of elements. ResultArray must be a dynamic array of a numeric data type (typically Variant or Long). If Array1 and Array2 are numeric types, comparison is done with the ">" and "<" operators. If Array1 and Array2 are string arrays, comparison is done with StrComp and the text-comparison mode (case-sensitive or case-insensitive) is determined by the CompareMode parameter.

ConcatenateArrays

```
Public Function ConcatenateArrays(ResultArray As Variant, ArrayToAppend As Variant, _
        Optional NoCompatabilityCheck As Boolean = False) As Boolean
```

This function appends the the ArrayToAppend array to the end of ResultArray. The Result array, which will hold its original values and the values of ArrayToAppend at the end of the array, must be a dynamic array. The Result array will be resized to hold its original data plus the data in the ArrayToAppend array. ArrayToAppend may be either a static or dynamic array. Either or both the Result array and the ArrayToAppend array may be unallocated. If the Result array is unallocated, and ArrayToAppend is allocated, the Result array is set to the same size as ArrayToAppend, and the LBound and UBound of the Result array will be the same as ArrayToAppend. If the ArrayToAppend is unallocated, the Result array is left intact and the function terminates. If both arrays are unallocated, no action is taken, the arrays remain unchanged, and the procedure terminates.

By default, ConcatenateArrays ensures that the data types of ResultArray and the ArrayToAppend array are equal or compatible. A destination element is compatible with a source element if the value of source element can be stored in the destination element without loss of precision or an overflow. For example, a destination Long is compatible with a source Integer because you can store an Integer in a Long with no loss of information or overflow. A destination Long is not compatible with a source Double because a Double cannot be stored in a Long without loss of information (the decimal portion will be lost) or possibility of overflow. The function `AreDataTypesCompatible` is used to test compatible data types. You can skip the compatibility test by setting the NoCompatibilityCheck parameter to True. Note, though, that this may cause information to be lost (decimal places may be lost when copying a Single or Double to an Integer or Long) or you may encounter an overflow condition, in which case that element of the destination array will be set to 0. If an overflow error occurs, the procedure ignores it and sets the destination array element to 0.

CopyArray

```
Public Function CopyArray(DestinationArray As Variant, SourceArray As Variant, _
        Optional NoCompatabilityCheck As Boolean = False) As Boolean
```

This function copies SourceArray to DestinationArray. Unfortunately, VBA does not allow you to copy one array to another with a simple assignment statement. You must copy the array element by element. If DestinationArray is dynamic, it is resized to hold all of the values in SourceArray. The DestinationArray will have the same lower and upper bounds of the SourceArray. If the DestinationArray is static, and the Source array has more elements than the Destination array, only the left-most elements of SourceArray are copied to fill DestinationArray. If DestinationArray is static and the SourceArray has fewer elements that the Destination array, the right-most elements of Destination array are left intact. The DestinationArray is not resized to match the SourceArray. If the SourceArray is empty (unallocated), the Destination array is left intact. If both the SourceArray and the DestinationArray are unallocated, the function exits and neither array is modified.

By default, CopyArray ensures that the data types of the Source and Destination arrays are equal or compatible. A destination element is compatible with a source element if the value of source element can be stored in the destination element without loss of precision or an overflow. For example, a destination Long is compatible with a source Integer because you can store an Integer in a Long with no loss of information or overflow. A destination Long is not compatible with a source Double because a Double cannot be stored in a Long without loss of information (the decimal portion will be lost) or possibility of overflow. The function `AreDataTypesCompatible` is used to test compatible data types. You can skip the compatibility test by setting the NoCompatibilityCheck parameter to True. Note, though, that this may cause information to be lost (decimal places may be lost when copying a Single or Double to an Integer or Long) or you may encounter an overflow condition, in which case that element of the destination array will be set to 0. If an overflow error occurs, the procedure ignores it and sets the destination array element to 0.

CopyArraySubSetToArray

```
Public Function CopyArraySubSetToArray(InputArray As Variant, ResultArray As Variant, _
    FirstElementToCopy As Long, LastElementToCopy As Long, DestinationElement As Long) As Boolean
```

This function copies a subset of InputArray to a location in ResultArray. The elements between FirstElementToCopy and LastElementToCopy (inclusive) of InputArray are copied to ResultArray, starting at DestinationElement. Existing data in ResultArray is overwritten. If ResultArray is not large enough to store the new data, it is resized appropriately if it is a dynamic array. If ResultArray is a static array and is not large enough to hold the new data, an error occurs and the function returns False. Both InputArray and ResultArray may be dynamic arrays, but InputArray must be allocated. ResultArray may be unallocated. If ResultArray is unallocated, it is resized with an LBound of 1 and a UBound of DestinationElement + NumElementsToCopy - 1. The elements to the left of DestinationElement are default values for the arrays data type (0, vbNullString, Empty, or Nothing). No type checking is done when copying the elements from one array to another. If InputArray is not compatible with ResultArray, no error is raised and the value in the ResultArray will be the default value for the data type of the array (0, vbNullString, Empty, or Nothing).

CopyNonNothingObjectToArray

```
Public Function CopyNonNothingObjectsToArray(ByRef SourceArray As Variant, _
    ByRef ResultArray As Variant, Optional NoAlerts As Boolean = False) As Boolean
```

This function copies all objects in SourceArray that are not Nothing to a new ResultArray. ResultArray must be declared as a dynamic array of Objects or Variants. SourceArray must contain all object-type variables (although the object types may be mixed -- the array may contain more than one type of object) or Nothing objects. An error will occur if a non-object variable is found in SourceArray.

DataTypeOfArray

```
Public Function DataTypeOfArray(Arr As Variant) As VbVarType
```

This function returns the data type (a `VbVarType` value) of the specified array. If the specified array is a simple array,
either single- or multi-dimensional, the function returns its data type. The specified array may be unallocated. If the variable passed in to `DataTypeOfArray` is not an array, the function returns -1. If the specified array is an array of arrays, the result is `vbArray`. For example,

```
Dim V(1 to 5) As String
Dim R As VbVarType
R = DataTypeOfArray(V)   ' returns vbString = 8
```

DeleteArrayElement

```
Public Function DeleteArrayElement(InputArray As Variant, ElementNumber As Long, _
    Optional ResizeDynamic As Boolean = False) As Boolean
```

This function deletes the specified element from the InputArray, shifting everything to the right of the deleted element one position to the left. The last element of the array is set to the appropriate default value (0, vbNullString, Empty, or Nothing) depending on the type of data in the array. The data type is determined by the last element in the array. By default, the size of the array is not changed. If the ResizeDynamic parameter is True and InputArray is a dynamic array, it will be resized down by one to remove the last element of the array. If the input array has one element, it is Erased.

ExpandArray

```
Function ExpandArray(Arr As Variant, WhichDim As Long, AdditionalElements As Long, _
        FillValue As Variant) As Variant
```

This function expands a two-dimensional array in either dimension. It returns an array with additional rows or columns. Rows are added at the bottom or the array, and columns are added at the right of the array. Arr is the original array. This array is not modified in any way. WhichDim indicates whether to add additional rows (WhichDim = 1) or additional columns (WhichDim = 2). AdditionalElements indicates the number of additional rows or columns to add to Arr. The new array elements are initialized with the value in FillValue. The function returns NULL if an error occurred. This function may be nested to add both rows and columns. The following code adds 3 rows and then 4 columns to the array A and puts the result array in C.

```
Dim A()
Dim B()
Dim C()
''''''''''''''''''''''''''''''''''''''''''''''''''
' Redim A, B, and C, and give them some values here.
''''''''''''''''''''''''''''''''''''''''''''''''''
C = ExpandArray(ExpandArray(Arr:=A, WhichDim:=1, AdditionalElements:=3, FillValue:="R"), _
    WhichDim:=2, AdditionalElements:=4, FillValue:="C")
```

FirstNonEmptyStringIndexInArray

```
Public Function FirstNonEmptyStringIndexInArray(InputArray As Variant) As Long
```

This function returns the index number of the first entry in an array of strings of an element that is no equal to vbNullString. This is useful when working with arrays of strings that have been sorted in ascending order, which places vbNullString entries at the beginning of the array. In general usages, The InputArray will be sorted in ascending order. For example,

```
Dim A(1 To 4) As String
Dim R As Long
A(1) = vbNullString
A(2) = vbNullString
A(3) = "A"
A(4) = "B"
R = FirstNonEmptyStringIndexInArray(InputArray:=A)
' R = 3, the first element that is not an empty string
Debug.Print "FirstNonEmptyStringIndexInArray", CStr(R)
```

GetColumn

```
Function GetColumn(Arr As Variant, ResultArr As Variant, ColumnNumber As Long) As Boolean
```

This function populates ResultArr with a one-dimensional array that is the column specified by ColumnNumber of the input array Arr. ResultArr must be a dynamic array. The existing contents of ResultArr are destroyed.

GetRow

```
Function GetRow(Arr As Variant, ResultArr As Variant, RowNumber As Long) As Boolean
```

This function populates ResultArr with a one-dimensional array that is the row specified by RowNumber of the input array Arr. ResultArr must be a dynamic array. The existing contents of ResultArr are destroyed.

InsertElementIntoAnArray

```
Public Function InsertElementIntoArray(InputArray As Variant, Index As Long, _
    Value As Variant) As Boolean
```

This function inserts the value Value at location Index in InputArray. InputArray must be a single-dimensional dynamic array. It will be resized to make room for the new data element. To insert an element at the end of the array, set Index to UBound(Array)+1.

IsArrayAllDefault

```
Public Function IsArrayAllDefault(InputArray As Variant) As Boolean
```

This function returns TRUE or FALSE indicating whether all the elements in the array have the default value for the particular data type. Depending on the data type of the array, the default value may be vbNullString, 0, Empty, or Nothing.

IsArrayAllNumeric

```
Public Function IsArrayAllNumeric(Arr As Variant, _
    Optional AllowNumericStrings As Boolean = False) As Boolean
```

This function returns TRUE or FALSE indicating whether all the elements in the array are numeric. By default Strings are not considered numeric, even if they contain numeric values. To allow numeric strings, set the AllowNumericStrings parameter to True.

IsArrayAllocated

```
Public Function IsArrayAllocated(Arr As Variant) As Boolean
```

This function returns TRUE or FALSE indicating whether the specified array is allocated (not empty). Returns TRUE of the array is a static array or a dynamic that has been allocated with a Redim statement. Returns FALSE if the array is a dynamic array that has not yet been sized with ReDim or that has been deallocated with the Erase statement. This function is basically the opposite of ArrayIsEmpty. For example,

```
Dim V() As Long
Dim R As Boolean
R = IsArrayAllocated(V)  ' returns false
ReDim V(1 To 10)
R = IsArrayAllocated(V)  ' returns true
```

IsArrayDynamic

```
Public Function IsArrayDynamic(ByRef Arr As Variant) As Boolean
```

This function returns TRUE or FALSE indicating whether the specified array is dynamic. Returns TRUE if the array is dynamic, or FALSE if the array is static. For example,

```
Dim DynArray() As Long
Dim StatArray(1 To 3) As Long
Dim B As Boolean
B = IsArrayDynamic(DynArray)   ' returns True
B = IsArrayDynamic(StatArray)  ' returns False
```

IsArrayEmpty

```
Public Function IsArrayEmpty(Arr As Variant) As Boolean
```

This function returns TRUE or FALSE indicating whether the specified array is empty (not allocated). This function is basically the opposite of IsArrayAllocated.

```
Dim DynArray() As Long
Dim R As Boolean
R = IsArrayEmpty(DynArray)  ' returns true
ReDim V(1 To 10)
R = IsArrayEmpty(DynArray)  ' returns false
```

IsArrayObjects

```
Public Function IsArrayObjects(InputArray As Variant, _
    Optional AllowNothing As Boolean = True) As Boolean
```

This function returns TRUE or FALSE indicating whether the specified array contains all Object variables. The objects may be of mixed type. By default, the function allows Nothing objects. That is, an object that is Nothing is still considered an object. To return False if an object is Nothing, set the AllowNothing parameter to False.

IsArraySorted

```
Public Function IsArraySorted(TestArray As Variant, _
    Optional Descending As Boolean = False) As Variant
```

This function returns TRUE or FALSE indicating whether the array TestArray is in sorted order (Ascending or Descending, depending on the value of the Descending parameter). It will return NULL if an error occurred. TestArray must be a single-dimensional allocated array. Since sorting is an expensive operation, especially so with large array of Strings or Variants, you can call this function to determine if the array is already sorted before calling upon the Sort procedures. If this function returns True, you don't need to resort the array.

IsNumericDataType

```
Public Function IsNumericDataType(TestVar As Variant) As Boolean
```

This indicates whether a variable is a numeric data type (Long, Integer, Double, Single, Currency, or Decimal). If the input is an array, it tests the first element of the array (note that in an array of variants, subsequent elements may not be numeric). For variable arrays, use `IsVariantArrayNumeric`.

IsVariantArrayConsistent

```
Public Function IsVariantArrayConsistent(Arr As Variant) As Boolean
```

This returns TRUE if all the data types in an array of Varaints all have the same data type. Otherwise, it returns False. If the array consists of Object type variables, objects that are Nothing are skipped. The function will return True if all non-object variables are the same type.

IsVariantArrayNumeric

```
Public Function IsVariantArrayNumeric(TestArray As Variant) As Boolean
```

This function returns TRUE or FALSE if an array of variants contains all numeric data types. The data types need not be the same. You can have a mix of Integers, Longs, Singles, and Doubles, and Emptys. As long as all the data types are numeric (as determined by the `IsNumericDataType` function), the result will be false. The function will return FALSE if the data types or not all numeric, or if the passed-in parameter is not an array or is an unallocated array. This procedure will work with multi-dimensional arrays.

MoveEmptyStringsToEndOfArray

```
Public Function MoveEmptyStringsToEndOfArray(InputArray As Variant) As Boolean
```

This moves empty strings at the beginning of the array to the end of the array, shifting elements of the array to the left. This is useful when dealing with a sorted array of text strings in which empty strings are placed at the beginning of the array. For example:

```
Dim S(1 to 4) As String
Dim N As Long
S(1) = vbNullString
S(2) = vbNullString
S(3) = "abc"
S(4) = "def"
N = MoveEmptyStringsToEndOfArray(S)
' resulting array:
For N = LBound(S) To UBound(S)
    If S(N) = vbNullString Then
        Debug.Print CStr(N),"Is vbNullString"
    Else
        Debug.Print CStr(N), S(N)
    End If
Next N
```

NumberOfDimensions

```
Public Function NumberOfArrayDimensions(Arr As Variant) As Integer
```

This function returns the number of dimensions of the specified array. If the array is a dynamic unallocated array, it returns 0.

```
Dim V(1 to 10, 1 to 5) As Long
Dim N As Long
N = NumberOfDimensions(V)  ' returns 2
```

NumElements

```
Public Function NumElements(Arr As Variant, Optional Dimension = 1) As Long
```

This function returns the number of elements in the specified dimension of the specified array. It returns 0 if an error condition exists (e.g., an unallocated array).

```
Dim V(1 to 10) As Long
Dim Dimension As Long
Dim N As Long
Dimension = 1
N = NumElements(V, Dimension)  ' returns 10
```

SetVariableToDefault

```
Public Sub SetVariableToDefault(ByRef Variable As Variant)
```

This procedure sets the Variable argument to the default value appropriate for its data type. This default may be 0, vbNullString, Empty, or Nothing. Note that it cannot reset a User-Defined Type. You can easily set a user defined type back to its default state by declaring a second variable of that type, e.g., `Dim DefaultType As MyType` and letting the elements take their default value. Then use `LSet` to set another instance of your UDT to `DefaultType`:

```
Public Type MyType
    X As Long
    Y As Long
    S As String
End Type

Dim DefaultType As MyType
Dim DataT As MyType
DataT.X = 1
DataT.Y = 2
DataT.S = "Test"
' set variables of T back to defaults.
LSet DataT  = DefaultType
```

Sorting An Array

See the Sorting Arrays With QSort page.

Sorting An Array Of Objects

See the Sorting Arrays Of Objects page.

SwapArrayRows and SwapArrayColumns

```
    Function SwapArrayColumns(Arr As Variant, Col1 As Long, Col2 As Long) As Variant

    Function SwapArrayRows(Arr As Variant, Row1 As Long, Row2 As Long) As Variant
```

These functions take in an array Arr and return a copy of the array with the specified rows or columns swapped.

ResetVariantArrayToDefaults

```
Public Function ResetVariantArrayToDefaults(InputArray As Variant) As Boolean
```

This function resets all the elements of an array of Variants to the appropriate default value (0, vbNullString, Empty, or Nothing). The array may consist of several different data types (e.g., some Longs, some Objects, some Strings, etc) and each element will be reset to the appropriate default value.

ReverseArrayInPlace

```
Public Function ReverseArrayInPlace(InputArray As Variant, _
    Optional NoAlerts As Boolean = False) As Boolean
```

This sub reverses the order of an array. It does the reversal in place. That is, the array variable in the calling procedure is reversed. The input array must be a single-dimensional array. The function returns True if the array was successfully reversed, or False if an error occurred.

```
Dim V(1 to 10) As String
Dim B As Boolean
' load V with some values
B = ReverseArrayInPlace(V)
```

ReverseArrayOfObjectsInPlace

```
Public Function ReverseArrayOfObjectsInPlace(InputArray As Variant, _
    Optional NoAlerts As Boolean = False) As Boolean
```

This sub reverses the order of an array. It does the reversal in place. That is, the array variable in the calling procedure is reversed. The function returns True or False indicating whether the array was successfully reversed. An error will occur if an array element is not an object (Nothing objects are allowed).

```
Dim V(1 to 10) As Object
Dim B As Boolean
' load V with some objects
B = ReverseArrayOfObjectsInPlace(V)
```

SetObjectArrayToNothing

```
Public Function SetObjectArrayToNothing(InputArray As Variant) As Boolean
```

This function sets all the elements of the specified array to Nothing. The InputArray must be declared as an array of objects, either a specific object type or a generic Object, or as Variants. An error occurs if an element in the array is not an object or Nothing. The array is not resized -- it remains the same size. Use this function instead of Erase when working with arrays of variants because Erase will set each element in the array to Empty, not Nothing and the element will cease to be considered an Object.

SetVariableToDefault

```
Public Sub SetVariableToDefault(ByRef Variable As Variant)
```

This procedure sets Variable to the appropriate default value for its data type. This default value will be 0, vbNullString, Empty, or Nothing depending on the data type of Variable.

TransposeArray

```
Public Function TransposeArray(InputArr As Variant, OutputArr As Variant) As Boolean
```

This procedure transposes a two dimensional array, creating a result array with the number of rows equal to the number of columns in the input array, and the number of columns equal to the number of rows in the input array. The LBounds and UBounds are preserved. The OutputArr must be a dynamic array. It will be Erased and Redim'd, so any existing content will be destroyed.

VectorsToArray

```
Public Function VectorsToArray(Arr As Variant, ParamArray Vectors()) As Boolean
```

This procedure takes any number of single dimensional arrays and combines them into a single two-dimensional array. The input arrays are in the ParamArray Vectors() parameter, and the array into which they will be placed is specified by Arr. Arr MUST be a dynamic array, and its data type must be compatible with all the elements in all the vectors. Arr is Erased and then Redim'd, so any existing content is destroyed. Each array in Vectors must be a single-dimensional allocated array. If a Vector is an unallocated array, the function will exit with a result of False.

Each array in Vectors is one row of Arr. The number of rows in Arr is the number of Vectors passed in. Each row of Arr is one vector. The number of columns is the *maximum* of the sizes of the Vectors. Since Arr is Erased, unused elements in Arr remain at the default value ofr the data type of Arr (the default value is either 0, vbNullString, or Empty, depending on how Arr was allocated). The elements of each vector must be simple data types. Objects, arrays, and user-defined types are not allowed. Both the rows and columns of Arr are 0-based, regardless of the original setting of Arr, the LBounds of each vector, and the Option Base statement. The vectors may be of different sizes and have different LBounds.

The VBA Code For The Functions

You can download a bas module file containing these function here or a complete workbook containing the functions and demonstration procedures here. Please read the comments within each procedure. They include important information about how the function works under various conditions. Many of these functions call upon one another, so it is recommended that you include the entire module within your project.

```
Option Explicit
Option Compare Text
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' modArraySupport
' By Chip Pearson, chip@cpearson.com, www.cpearson.com
'
' This module contains procedures that provide information about and manipulate
' VB/VBA arrays.
'
' For details on these functions, see www.cpearson.com/excel/VBAArrays.htm
'
' This module contains the following functions:
'       AreDataTypesCompatible
'       ChangeBoundsOfArray
'       CompareArrays
'       ConcatenateArrays
'       CopyArray
'       CopyArraySubSetToArray
'       CopyNonNothingObjectsToArray
'       DataTypeOfArray
'       DeleteArrayElement
'       FirstNonEmptyStringIndexInArray
'       InsertElementIntoArray
'       IsArrayAllDefault
'       IsArrayAllNumeric
'       IsArrayAllocated
'       IsArrayDynamic
'       IsArrayEmpty
'       IsArrayObjects
'       IsNumericDataType
'       IsVariantArrayConsistent
'       IsVariantArrayNumeric
'       MoveEmptyStringsToEndOfArray
'       NumberOfArrayDimensions
'       NumElements
'       ResetVariantArrayToDefaults
'       ReverseArrayInPlace
'       ReverseArrayOfObjectsInPlace
'       SetObjectArrayToNothing
'       SetVariableToDefault
'       TransposeArray
'       VectorsToArray
'
' Function documentation is in each function.
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''
' Error Number Constants
'''''''''''''''''''''''''''
Public Const C_ERR_NO_ERROR = 0&
Public Const C_ERR_SUBSCRIPT_OUT_OF_RANGE = 9&
Public Const C_ERR_ARRAY_IS_FIXED_OR_LOCKED = 10&

Public Function ChangeBoundsOfArray(InputArr As Variant, _
    NewLowerBound As Long, NewUpperBound) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' ChangeBoundsOfArray
' This function changes the lower and upper bounds of the specified
' array. InputArr MUST be a single-dimensional, dynamic, allocated array.
' If the new size of the array (NewUpperBound - NewLowerBound + 1)
' is greater than the original array, the unused elements on
' right side of the new array are the default values for the data type
' of the array. If the new size is less than the original size,
' only the first (left-most) N elements are included in the new array.
' The elements of the array may be simple variables (Strings, Longs, etc)
' Objects, or Arrays. User-Defined Types are not supported.
'
' The function returns True if successful, False otherwise.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''


Dim TempArr() As Variant
Dim InNdx As Long
Dim OutNdx As Long
Dim TempNdx As Long
Dim FirstIsObject As Boolean

''''''''''''''''''''''''''''''''''''
' Ensure we have an array.
''''''''''''''''''''''''''''''''''''
If IsArray(InputArr) = False Then
    ChangeBoundsOfArray = False
    Exit Function
End If
''''''''''''''''''''''''''''''''''''
' Ensure the array is dynamic.
''''''''''''''''''''''''''''''''''''
If IsArrayDynamic(InputArr) = False Then
    ChangeBoundsOfArray = False
    Exit Function
End If
''''''''''''''''''''''''''''''''''''
' Ensure the array is allocated.
''''''''''''''''''''''''''''''''''''
If IsArrayAllocated(InputArr) = False Then
    ChangeBoundsOfArray = False
    Exit Function
End If
'''''''''''''''''''''''''''''''''''''''''''
' Ensure the NewLowerBound > NewUpperBound.
'''''''''''''''''''''''''''''''''''''''''''
If NewLowerBound > NewUpperBound Then
    ChangeBoundsOfArray = False
    Exit Function
End If
''''''''''''''''''''''''''''''''''''''''''''
' Ensure Arr is a single dimensional array.
'''''''''''''''''''''''''''''''''''''''''''
If NumberOfArrayDimensions(InputArr) <> 1 Then
    ChangeBoundsOfArray = False
    Exit Function
End If

''''''''''''''''''''''''''''''''''''''''''''''''''''
' We need to save the IsObject status of the first
' element of the InputArr to properly handle
' the Empty variables is we are making the array
' larger than it was before.
'''''''''''''''''''''''''''''''''''''''''''''''''''
FirstIsObject = IsObject(InputArr(LBound(InputArr)))


''''''''''''''''''''''''''''''''''''''''''''
' Resize TempArr and save the values in
' InputArr in TempArr. TempArr will have
' an LBound of 1 and a UBound of the size
' of (NewUpperBound - NewLowerBound +1)
'''''''''''''''''''''''''''''''''''''''''''
ReDim TempArr(1 To (NewUpperBound - NewLowerBound + 1))
'''''''''''''''''''''''''''''''''''''''''''
' Load up TempArr
'''''''''''''''''''''''''''''''''''''''''''
TempNdx = 0
For InNdx = LBound(InputArr) To UBound(InputArr)
    TempNdx = TempNdx + 1
    If TempNdx > UBound(TempArr) Then
        Exit For
    End If

    If (IsObject(InputArr(InNdx)) = True) Then
        If InputArr(InNdx) Is Nothing Then
            Set TempArr(TempNdx) = Nothing
        Else
            Set TempArr(TempNdx) = InputArr(InNdx)
        End If
    Else
        TempArr(TempNdx) = InputArr(InNdx)
    End If
Next InNdx

''''''''''''''''''''''''''''''''''''
' Now, Erase InputArr, resize it to the
' new bounds, and load up the values from
' TempArr to the new InputArr.
''''''''''''''''''''''''''''''''''''
Erase InputArr
ReDim InputArr(NewLowerBound To NewUpperBound)
OutNdx = LBound(InputArr)
For TempNdx = LBound(TempArr) To UBound(TempArr)
    If OutNdx <= UBound(InputArr) Then
        If IsObject(TempArr(TempNdx)) = True Then
            Set InputArr(OutNdx) = TempArr(TempNdx)
        Else
            If FirstIsObject = True Then
                If IsEmpty(TempArr(TempNdx)) = True Then
                    Set InputArr(OutNdx) = Nothing
                Else
                    Set InputArr(OutNdx) = TempArr(TempNdx)
                End If
            Else
                InputArr(OutNdx) = TempArr(TempNdx)
            End If
        End If
    Else
        Exit For
    End If
    OutNdx = OutNdx + 1
Next TempNdx

'''''''''''''''''''''''''''''
' Success -- Return True
'''''''''''''''''''''''''''''
```

```
ChangeBoundsOfArray = True

End Function



Public Function CompareArrays(Array1 As Variant, Array2 As Variant, _
    ResultArray As Variant, Optional CompareMode As VbCompareMethod = vbTextCompare) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' CompareArrays
' This function compares two arrays, Array1 and Array2, element by element, and puts the results of
' the comparisons in ResultArray. Each element of ResultArray will be -1, 0, or +1. A -1 indicates that
' the element in Array1 was less than the corresponding element in Array2. A 0 indicates that the
' elements are equal, and +1 indicates that the element in Array1 is greater than Array2. Both
' Array1 and Array2 must be allocated single-dimensional arrays, and ResultArray must be dynamic array
' of a numeric data type (typically Longs). Array1 and Array2 must contain the same number of elements,
' and have the same lower bound. The LBound of ResultArray will be the same as the data arrays.
'
' An error will occur if Array1 or Array2 contains an Object or User Defined Type.
'
' When comparing elements, the procedure does the following:
' If both elements are numeric data types, they are compared arithmetically.

' If one element is a numeric data type and the other is a string and that string is numeric,
' then both elements are converted to Doubles and compared arithmetically. If the string is not
' numeric, both elements are converted to strings and compared using StrComp, with the
' compare mode set by CompareMode.
'
' If both elements are numeric strings, they are converted to Doubles and compared arithmetically.
'
' If either element is not a numeric string, the elements are converted and compared with StrComp.
'
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim Ndx1 As Long
Dim Ndx2 As Long
Dim ResNdx As Long
Dim S1 As String
Dim S2 As String
Dim D1 As Double
Dim D2 As Double
Dim Done As Boolean
Dim Compare As VbCompareMethod
Dim LB As Long

'''''''''''''''''''''''''''''''''''
' Set the default return value.
'''''''''''''''''''''''''''''''''''
CompareArrays = False

'''''''''''''''''''''''''''''''''''
' Ensure we have a Compare mode
' value.
'''''''''''''''''''''''''''''''''''
If CompareMode = vbBinaryCompare Then
    Compare = vbBinaryCompare
Else
    Compare = vbTextCompare
End If


'''''''''''''''''''''''''''''''''''
' Ensure we have arrays.
'''''''''''''''''''''''''''''''''''
If IsArray(Array1) = False Then
    Exit Function
End If
If IsArray(Array2) = False Then
    Exit Function
End If
If IsArray(ResultArray) = False Then
    Exit Function
End If

''''''''''''''''''''''''''''''''''
' Ensure ResultArray is dynamic
''''''''''''''''''''''''''''''''''
If IsArrayDynamic(Arr:=ResultArray) = False Then
    Exit Function
End If

'''''''''''''''''''''''''''''''''''''''''
' Ensure the arrays are single-dimensional.
'''''''''''''''''''''''''''''''''''''''''
If NumberOfArrayDimensions(Arr:=Array1) <> 1 Then
    Exit Function
End If
If NumberOfArrayDimensions(Arr:=Array2) <> 1 Then
    Exit Function
End If
If NumberOfArrayDimensions(Arr:=Array1) > 1 Then 'allow 0 indicating non-allocated array
    Exit Function
End If

'''''''''''''''''''''''''''''''''''''''''
' Ensure the LBounds are the same
'''''''''''''''''''''''''''''''''''''''''
If LBound(Array1) <> LBound(Array2) Then
    Exit Function
```

```
End If

''''''''''''''''''''''''''''''''''''''''''
' Ensure the arrays are the same size.
''''''''''''''''''''''''''''''''''''''''''
If (UBound(Array1) - LBound(Array1)) <> (UBound(Array2) - LBound(Array2)) Then
    Exit Function
End If

''''''''''''''''''''''''''''''''''''''''''''''
' Redim ResultArray to the numbr of elements
' in Array1.
''''''''''''''''''''''''''''''''''''''''''''''
ReDim ResultArray(LBound(Array1) To UBound(Array1))

Ndx1 = LBound(Array1)
Ndx2 = LBound(Array2)

''''''''''''''''''''''''''''''''''''''''''''''
' Scan each array to see if it contains objects
' or User-Defined Types. If found, exit with
' False.
''''''''''''''''''''''''''''''''''''''''''''''
For Ndx1 = LBound(Array1) To UBound(Array1)
    If IsObject(Array1(Ndx1)) = True Then
        Exit Function
    End If
    If VarType(Array1(Ndx1)) >= vbArray Then
        Exit Function
    End If
    If VarType(Array1(Ndx1)) = vbUserDefinedType Then
        Exit Function
    End If
Next Ndx1

For Ndx1 = LBound(Array2) To UBound(Array2)
    If IsObject(Array2(Ndx1)) = True Then
        Exit Function
    End If
    If VarType(Array2(Ndx1)) >= vbArray Then
        Exit Function
    End If
    If VarType(Array2(Ndx1)) = vbUserDefinedType Then
        Exit Function
    End If
Next Ndx1

Ndx1 = LBound(Array1)
Ndx2 = Ndx1
ResNdx = LBound(ResultArray)
Done = False
Do Until Done = True
''''''''''''''''''''''''''''''''''''
' Loop until we reach the end of
' the array.
''''''''''''''''''''''''''''''''''''
    If IsNumeric(Array1(Ndx1)) = True And IsNumeric(Array2(Ndx2)) Then
        D1 = CDbl(Array1(Ndx1))
        D2 = CDbl(Array2(Ndx2))
        If D1 = D2 Then
            ResultArray(ResNdx) = 0
        ElseIf D1 < D2 Then
            ResultArray(ResNdx) = -1
        Else
            ResultArray(ResNdx) = 1
        End If
    Else
        S1 = CStr(Array1(Ndx1))
        S2 = CStr(Array2(Ndx1))
        ResultArray(ResNdx) = StrComp(S1, S2, Compare)
    End If

    ResNdx = ResNdx + 1
    Ndx1 = Ndx1 + 1
    Ndx2 = Ndx2 + 1
    ''''''''''''''''''''''''''''''''''''''''
    ' If Ndx1 is greater than UBound(Array1)
    ' we've hit the end of the arrays.
    ''''''''''''''''''''''''''''''''''''''''
    If Ndx1 > UBound(Array1) Then
        Done = True
    End If
Loop

CompareArrays = True
End Function



Public Function ConcatenateArrays(ResultArray As Variant, ArrayToAppend As Variant, _
        Optional NoCompatabilityCheck As Boolean = False) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' ConcatenateArrays
' This function appends ArrayToAppend to the end of ResultArray, increasing the size of ResultArray
' as needed. ResultArray must be a dynamic array, but it need not be allocated. ArrayToAppend
' may be either static or dynamic, and if dynamic it may be unallocted. If ArrayToAppend is
' unallocated, ResultArray is left unchanged.
'
' The data types of ResultArray and ArrayToAppend must be either the same data type or
' compatible numeric types. A compatible numeric type is a type that will not cause a loss of
' precision or cause an overflow. For example, ReturnArray may be Longs, and ArrayToAppend amy
' by Longs or Integers, but not Single or Doubles because information might be lost when
' converting from Double to Long (the decimal portion would be lost). To skip the compatability
' check and allow any variable type in ResultArray and ArrayToAppend, set the NoCompatabilityCheck
' parameter to True. If you do this, be aware that you may loose precision and you may will
' get an overflow error which will cause a result of 0 in that element of ResultArra.
'
' Both ReaultArray and ArrayToAppend must be one-dimensional arrays.
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim VTypeResult As VbVarType
Dim Ndx As Long
Dim Res As Long
Dim NumElementsToAdd As Long
Dim AppendNdx As Long
Dim VTypeAppend As VbVarType
Dim ResultLB As Long
Dim ResultUB As Long
Dim ResultWasAllocated As Boolean

'''''''''''''''''''''''''''''''''
' Set the default result.
''''''''''''''''''''''''''''''''
ConcatenateArrays = False

'''''''''''''''''''''''''''''''''
' Ensure ResultArray is an array.
'''''''''''''''''''''''''''''''''
If IsArray(ResultArray) = False Then
    Exit Function
End If
''''''''''''''''''''''''''''''''''
' Ensure ArrayToAppend is an array.
''''''''''''''''''''''''''''''''''
If IsArray(ArrayToAppend) = False Then
    Exit Function
End If

''''''''''''''''''''''''''''''''''
' Ensure both arrays are single
' dimensional.
''''''''''''''''''''''''''''''''''
If NumberOfArrayDimensions(ResultArray) > 1 Then
    Exit Function
End If
If NumberOfArrayDimensions(ArrayToAppend) > 1 Then
    Exit Function
End If
'''''''''''''''''''''''''''''''''''
' Ensure ResultArray is dynamic.
'''''''''''''''''''''''''''''''''''
If IsArrayDynamic(Arr:=ResultArray) = False Then
    Exit Function
End If

''''''''''''''''''''''''''''''''''''
' Ensure ArrayToAppend is allocated.
' If ArrayToAppend is not allocated,
' we have nothing to append, so
' exit with a True result.
''''''''''''''''''''''''''''''''''''
If IsArrayAllocated(Arr:=ArrayToAppend) = False Then
    ConcatenateArrays = True
    Exit Function
End If


If NoCompatabilityCheck = False Then
    ''''''''''''''''''''''''''''''''''''''
    ' Ensure the array are compatible
    ' data types.
    ''''''''''''''''''''''''''''''''''''''
    If AreDataTypesCompatible(DestVar:=ResultArray, SourceVar:=ArrayToAppend) = False Then
        '''''''''''''''''''''''''''''''''''''''''''
        ' The arrays are not compatible data types.
        '''''''''''''''''''''''''''''''''''''''''''
        Exit Function
    End If

    ''''''''''''''''''''''''''''''''''''
    ' If one array is an array of
    ' objects, ensure the other contains
    ' all objects (or Nothing)
    ''''''''''''''''''''''''''''''''''''
    If VarType(ResultArray) - vbArray = vbObject Then
        If IsArrayAllocated(ArrayToAppend) = True Then
            For Ndx = LBound(ArrayToAppend) To UBound(ArrayToAppend)
                If IsObject(ArrayToAppend(Ndx)) = False Then
                    Exit Function
                End If
            Next Ndx
        End If
    End If
End If


'''''''''''''''''''''''''''''''''''''''
' Get the number of elements in
' ArrayToAppend
''''''''''''''''''''''''''''''''''''''
NumElementsToAdd = UBound(ArrayToAppend) - LBound(ArrayToAppend) + 1
''''''''''''''''''''''''''''''''''''''
' Get the bounds for resizing the
' ResultArray. If ResultArray is allocated
' use the LBound and UBound+1. If
' ResultArray is not allocated, use
' the LBound of ArrayToAppend for both
' the LBound and UBound of ResultArray.
''''''''''''''''''''''''''''''''''''''

If IsArrayAllocated(Arr:=ResultArray) = True Then
    ResultLB = LBound(ResultArray)
    ResultUB = UBound(ResultArray)
    ResultWasAllocated = True
    ReDim Preserve ResultArray(ResultLB To ResultUB + NumElementsToAdd)
Else
    ResultUB = UBound(ArrayToAppend)
    ResultWasAllocated = False
    ReDim ResultArray(LBound(ArrayToAppend) To UBound(ArrayToAppend))
End If

''''''''''''''''''''''''''''''''''''''
' Copy the data from ArrayToAppend to
' ResultArray.
''''''''''''''''''''''''''''''''''''''
If ResultWasAllocated = True Then
    ''''''''''''''''''''''''''''''''''''''''''
    ' If ResultArray was allocated, we
    ' have to put the data from ArrayToAppend
    ' at the end of the ResultArray.
    ''''''''''''''''''''''''''''''''''''''''''
    AppendNdx = LBound(ArrayToAppend)
    For Ndx = ResultUB + 1 To UBound(ResultArray)
        If IsObject(ArrayToAppend(AppendNdx)) = True Then
            Set ResultArray(Ndx) = ArrayToAppend(AppendNdx)
        Else
            ResultArray(Ndx) = ArrayToAppend(AppendNdx)
        End If
        AppendNdx = AppendNdx + 1
        If AppendNdx > UBound(ArrayToAppend) Then
            Exit For
        End If
    Next Ndx
Else
    ''''''''''''''''''''''''''''''''''''''''''''''
    ' If ResultArray was not allocated, we simply
    ' copy element by element from ArrayToAppend
    ' to ResultArray.
    ''''''''''''''''''''''''''''''''''''''''''''''
    For Ndx = LBound(ResultArray) To UBound(ResultArray)
        If IsObject(ArrayToAppend(Ndx)) = True Then
            Set ResultArray(Ndx) = ArrayToAppend(Ndx)
        Else
            ResultArray(Ndx) = ArrayToAppend(Ndx)
        End If
    Next Ndx

End If
'''''''''''''''''''''''
' Success. Return True.
'''''''''''''''''''''''
ConcatenateArrays = True

End Function

Public Function CopyArray(DestinationArray As Variant, SourceArray As Variant, _
        Optional NoCompatabilityCheck As Boolean = False) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' CopyArray
' This function copies the contents of SourceArray to the DestinationaArray. Both SourceArray
' and DestinationArray may be either static or dynamic and either or both may be unallocated.
'
' If DestinationArray is dynamic, it is resized to match SourceArray. The LBound and UBound
' of DestinationArray will be the same as SourceArray, and all elements of SourceArray will
' be copied to DestinationArray.
'
' If DestinationArray is static and has more elements than SourceArray, all of SourceArray
' is copied to DestinationArray and the right-most elements of DestinationArray are left
' intact.
'
' If DestinationArray is static and has fewer elements that SourceArray, only the left-most
' elements of SourceArray are copied to fill out DestinationArray.
'
' If SourceArray is an unallocated array, DestinationArray remains unchanged and the procedure
' terminates.
'
' If both SourceArray and DestinationArray are unallocated, no changes are made to either array
' and the procedure terminates.
'
' SourceArray may contain any type of data, including Objects and Objects that are Nothing
' (the procedure does not support arrays of User Defined Types since these cannot be coerced
' to Variants -- use classes instead of types).
'
' The function tests to ensure that the data types of the arrays are the same or are compatible.
' See the function AreDataTypesCompatible for information about compatible data types. To skip
' this compability checking, set the NoCompatabilityCheck parameter to True. Note that you may
' lose information during data conversion (e.g., losing decimal places when converting a Double
' to a Long) or you may get an overflow (storing a Long in an Integer) which will result in that
' element in DestinationArray having a value of 0.
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim VTypeSource As VbVarType
Dim VTypeDest As VbVarType
Dim SNdx As Long
Dim DNdx As Long


'''''''''''''''''''''''''''''''
' Set the default return value.
'''''''''''''''''''''''''''''''
CopyArray = False

''''''''''''''''''''''''''''''''''
' Ensure both DestinationArray and
' SourceArray are arrays.
''''''''''''''''''''''''''''''''''
If IsArray(DestinationArray) = False Then
    Exit Function
End If
If IsArray(SourceArray) = False Then
    Exit Function
End If

'''''''''''''''''''''''''''''''''''''
' Ensure DestinationArray and
' SourceArray are single-dimensional.
' 0 indicates an unallocated array,
' which is allowed.
'''''''''''''''''''''''''''''''''''''
If NumberOfArrayDimensions(Arr:=SourceArray) > 1 Then
    Exit Function
End If
If NumberOfArrayDimensions(Arr:=DestinationArray) > 1 Then
    Exit Function
End If

''''''''''''''''''''''''''''''''''''
' If SourceArray is not allocated,
' leave DestinationArray intact and
' return a result of True.
''''''''''''''''''''''''''''''''''''
If IsArrayAllocated(Arr:=SourceArray) = False Then
    CopyArray = True
    Exit Function
End If

If NoCompatabilityCheck = False Then
    ''''''''''''''''''''''''''''''''''''''
    ' Ensure both arrays are the same
    ' type or compatible data types. See
    ' the function AreDataTypesCompatible
    ' for information about compatible
    ' types.
    ''''''''''''''''''''''''''''''''''''''
    If AreDataTypesCompatible(DestVar:=DestinationArray, SourceVar:=SourceArray) = False Then
        CopyArray = False
        Exit Function
    End If
    ''''''''''''''''''''''''''''''''''''
    ' If one array is an array of
    ' objects, ensure the other contains
    ' all objects (or Nothing)
    ''''''''''''''''''''''''''''''''''''
    If VarType(DestinationArray) - vbArray = vbObject Then
        If IsArrayAllocated(SourceArray) = True Then
            For SNdx = LBound(SourceArray) To UBound(SourceArray)
                If IsObject(SourceArray(SNdx)) = False Then
                    Exit Function
                End If
            Next SNdx
        End If
    End If
End If

If IsArrayAllocated(Arr:=DestinationArray) = True Then
    If IsArrayAllocated(Arr:=SourceArray) = True Then
        '''''''''''''''''''''''''''''''''''''''''''''''''
        ' If both arrays are allocated, copy from
        ' SourceArray to DestinationArray. If
        ' SourceArray is smaller that DesetinationArray,
        ' the right-most elements of DestinationArray
        ' are left unchanged. If SourceArray is larger
        ' than DestinationArray, the right most elements
        ' of SourceArray are not copied.
        ''''''''''''''''''''''''''''''''''''''''''''''''''''
        DNdx = LBound(DestinationArray)
        On Error Resume Next
        For SNdx = LBound(SourceArray) To UBound(SourceArray)
            If IsObject(SourceArray(SNdx)) = True Then
                Set DestinationArray(DNdx) = SourceArray(DNdx)
            Else
                DestinationArray(DNdx) = SourceArray(DNdx)
            End If
            DNdx = DNdx + 1
            If DNdx > UBound(DestinationArray) Then
                Exit For
            End If
        Next SNdx
        On Error GoTo 0
    Else
        '''''''''''''''''''''''''''''''''''''''''''''''
        ' If SourceArray is not allocated, so we have
        ' nothing to copy. Exit with a result
        ' of True. Leave DestinationArray intact.
        '''''''''''''''''''''''''''''''''''''''''''''''
        CopyArray = True
        Exit Function
    End If

Else
    If IsArrayAllocated(Arr:=SourceArray) = True Then
        ''''''''''''''''''''''''''''''''''''''''''''''''''''
        ' If Destination array is not allocated and
        ' SourceArray is allocated, Redim DestinationArray
        ' to the same size as SourceArray and copy
        ' the elements from SourceArray to DestinationArray.
        ''''''''''''''''''''''''''''''''''''''''''''''''''''
        On Error Resume Next
        ReDim DestinationArray(LBound(SourceArray) To UBound(SourceArray))
        For SNdx = LBound(SourceArray) To UBound(SourceArray)
            If IsObject(SourceArray(SNdx)) = True Then
                Set DestinationArray(SNdx) = SourceArray(SNdx)
            Else
                DestinationArray(SNdx) = SourceArray(SNdx)
            End If
        Next SNdx
        On Error GoTo 0
    Else
        ''''''''''''''''''''''''''''''''''''''''''''''''''''
        ' If both SourceArray and DestinationArray are
        ' unallocated, we have nothing to copy (this condition
        ' is actually detected above, but included here
        ' for consistancy), so get out with a result of True.
        ''''''''''''''''''''''''''''''''''''''''''''''''''''
        CopyArray = True
        Exit Function
    End If
End If

'''''''''''''''''''''''
' Success. Return True.
'''''''''''''''''''''''
CopyArray = True

End Function



Public Function CopyArraySubSetToArray(InputArray As Variant, ResultArray As Variant, _
    FirstElementToCopy As Long, LastElementToCopy As Long, DestinationElement As Long) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' CopyArraySubSetToArray
' This function copies elements of InputArray to ResultArray. It takes the elements
' from FirstElementToCopy to LastElementToCopy (inclusive) from InputArray and
' copies them to ResultArray, starting at DestinationElement. Existing data in
' ResultArray will be overwrittten. If ResultArray is a dynamic array, it will
' be resized if needed. If ResultArray is a static array and it is not large
' enough to copy all the elements, no elements are copied and the function
' returns False.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim SrcNdx As Long
Dim DestNdx As Long
Dim NumElementsToCopy As Long

''''''''''''''''''''''''''''''''''''''''
' Set the default return value.
''''''''''''''''''''''''''''''''''''''''
CopyArraySubSetToArray = False

'''''''''''''''''''''''''''''''''''''''''
' Ensure InputArray and ResultArray are
' arrays.
'''''''''''''''''''''''''''''''''''''''''
If IsArray(InputArray) = False Then
    Exit Function
End If
If IsArray(ResultArray) = False Then
    Exit Function
End If
''''''''''''''''''''''''''''''''''''''''''
' Ensure InputArray is single dimensional.
''''''''''''''''''''''''''''''''''''''''''
If NumberOfArrayDimensions(Arr:=InputArray) <> 1 Then
    Exit Function
End If
''''''''''''''''''''''''''''''''''''''''''
' Ensure ResultArray is unallocated or
' single dimensional.
''''''''''''''''''''''''''''''''''''''''''
If NumberOfArrayDimensions(Arr:=ResultArray) > 1 Then
    Exit Function
End If

'''''''''''''''''''''''''''''''''''''''''''
' Ensure the bounds and indexes are valid.
'''''''''''''''''''''''''''''''''''''''''''
If FirstElementToCopy < LBound(InputArray) Then
```

```
    Exit Function
End If
If LastElementToCopy > UBound(InputArray) Then
   Exit Function
End If
If FirstElementToCopy > LastElementToCopy Then
    Exit Function
End If

''''''''''''''''''''''''''''''''''''''''''
' Calc the number of elements we'll copy
' from InputArray to ResultArray.
''''''''''''''''''''''''''''''''''''''''''
NumElementsToCopy = LastElementToCopy - FirstElementToCopy + 1

If IsArrayDynamic(Arr:=ResultArray) = False Then
    If (DestinationElement + NumElementsToCopy - 1) > UBound(ResultArray) Then
        ''''''''''''''''''''''''''''''''''''''''''''''
        ' ResultArray is static and can't be resized.
        ' There is not enough room in the array to
        ' copy all the data.
        ''''''''''''''''''''''''''''''''''''''''''''''
        Exit Function
    End If
Else
    ''''''''''''''''''''''''''''''''''''''''''''
    ' ResultArray is dynamic and can be resized.
    ' Test whether we need to resize the array,
    ' and resize it if required.
    '''''''''''''''''''''''''''''''''''''''''''''
    If IsArrayEmpty(Arr:=ResultArray) = True Then
        '''''''''''''''''''''''''''''''''''''''
        ' ResultArray is unallocated. Resize it
        ' to DestinationElement + NumElementsToCopy - 1.
        ' This provides empty elements to the left
        ' of the DestinationElement and room to
        ' copy NumElementsToCopy.
        '''''''''''''''''''''''''''''''''''''''''
        ReDim ResultArray(1 To DestinationElement + NumElementsToCopy - 1)
    Else
        '''''''''''''''''''''''''''''''''''''''''''''''''
        ' ResultArray is allocated. If there isn't room
        ' enough in ResultArray to hold NumElementsToCopy
        ' starting at DestinationElement, we need to
        ' resize the array.
        '''''''''''''''''''''''''''''''''''''''''''''''''
        If (DestinationElement + NumElementsToCopy - 1) > UBound(ResultArray) Then
            If DestinationElement + NumElementsToCopy > UBound(ResultArray) Then
                '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
                ' Resize the ResultArray.
                '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
                If NumElementsToCopy + DestinationElement > UBound(ResultArray) Then
                    ReDim Preserve ResultArray(LBound(ResultArray) To UBound(ResultArray) + DestinationElement - 1)
                Else
                    ReDim Preserve ResultArray(LBound(ResultArray) To UBound(ResultArray) + NumElementsToCopy)
                End If
            Else
                '''''''''''''''''''''''''''''''''''''''''''''
                ' Resize the array to hold NumElementsToCopy
                ' starting at DestinationElement.
                '''''''''''''''''''''''''''''''''''''''''''''
                ReDim Preserve ResultArray(LBound(ResultArray) To UBound(ResultArray) + NumElementsToCopy - DestinationElement + 2)
            End If
        Else
            '''''''''''''''''''''''''''''''''''''''''''''''''''''
            ' The ResultArray is large enough to hold
            ' NumberOfElementToCopy starting at DestinationElement.
            ' No need to resize the array.
            '''''''''''''''''''''''''''''''''''''''''''''''''''''''
        End If
    End If
End If


'''''''''''''''''''''''''''''''''''''''''''''''''''
' Copy the elements from InputArray to ResultArray
' Note that there is no type compatibility checking
' when copying the elements.
'''''''''''''''''''''''''''''''''''''''''''''''''''
DestNdx = DestinationElement
For SrcNdx = FirstElementToCopy To LastElementToCopy
    If IsObject(InputArray(SrcNdx)) = True Then
        Set ResultArray(DestNdx) = InputArray(DestNdx)
    Else
        On Error Resume Next
        ResultArray(DestNdx) = InputArray(SrcNdx)
        On Error GoTo 0
    End If
    DestNdx = DestNdx + 1
Next SrcNdx

CopyArraySubSetToArray = True

End Function



Public Function CopyNonNothingObjectsToArray(ByRef SourceArray As Variant, _
    ByRef ResultArray As Variant, Optional NoAlerts As Boolean = False) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
' CopyNonNothingObjectsToArray
' This function copies all objects that are not Nothing from SourceArray
' to ResultArray. ResultArray MUST be a dynamic array of type Object or Variant.
' E.g.,
'       Dim ResultArray() As Object ' Or
'       Dim ResultArray() as Variant
'
' ResultArray will be Erased and then resized to hold the non-Nothing elements
' from SourceArray. The LBound of ResultArray will be the same as the LBound
' of SourceArray, regardless of what its LBound was prior to calling this
' procedure.
'
' This function returns True if the operation was successful or False if an
' an error occurs. If an error occurs, a message box is displayed indicating
' the error. To suppress the message boxes, set the NoAlerts parameter to
' True.
'
' This function uses the following procedures. They are declared as Private
' procedures at the end of this module.
'       IsArrayDynamic
'       IsArrayEmpty
'       NumberOfArrayDimensions
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim ResNdx As Long
Dim InNdx  As Long

''''''''''''''''''''''''''''''''''
' Set the default return value.
''''''''''''''''''''''''''''''''''
CopyNonNothingObjectsToArray = False

''''''''''''''''''''''''''''''''''''
' Ensure SourceArray is an array.
''''''''''''''''''''''''''''''''''''
If IsArray(SourceArray) = False Then
    If NoAlerts = False Then
        MsgBox "SourceArray is not an array."
    End If
    Exit Function
End If
''''''''''''''''''''''''''''''''''''
' Ensure SourceArray is a single
' dimensional array.
''''''''''''''''''''''''''''''''''''
Select Case NumberOfArrayDimensions(Arr:=SourceArray)
    Case 0
        ''''''''''''''''''''''''''''''
        ' Unallocated dynamic array.
        ' Not Allowed.
        ''''''''''''''''''''''''''''''
        If NoAlerts = False Then
            MsgBox "SourceArray is an unallocated array."
        End If
        Exit Function

    Case 1
        ''''''''''''''''''''''''''''''
        ' Single-dimensional array.
        ' This is OK.
        ''''''''''''''''''''''''''''''
    Case Else
        ''''''''''''''''''''''''''''''
        ' Multi-dimensional array.
        ' This is not allowed.
        ''''''''''''''''''''''''''''''
        If NoAlerts = False Then
            MsgBox "SourceArray is a multi-dimensional array. This is not allowed."
        End If
        Exit Function
End Select
''''''''''''''''''''''''''''''''''''
' Ensure ResultArray is an array.
''''''''''''''''''''''''''''''''''''
If IsArray(ResultArray) = False Then
    If NoAlerts = False Then
        MsgBox "ResultArray is not an array."
    End If
    Exit Function
End If
''''''''''''''''''''''''''''''''''''
' Ensure ResultArray is an dynamic.
''''''''''''''''''''''''''''''''''''
If IsArrayDynamic(Arr:=ResultArray) = False Then
    If NoAlerts = False Then
        MsgBox "ResultArray is not a dynamic array."
    End If
    Exit Function
End If
''''''''''''''''''''''''''''''''''''
' Ensure ResultArray is a single
' dimensional array.
''''''''''''''''''''''''''''''''''''
Select Case NumberOfArrayDimensions(Arr:=ResultArray)
    Case 0
        ''''''''''''''''''''''''''''''
        ' Unallocated dynamic array.
        ' This is OK.
        ''''''''''''''''''''''''''''''
    Case 1
```

```
        ''''''''''''''''''''''''''''''
        ' Single-dimensional array.
        ' This is OK.
        ''''''''''''''''''''''''''''''
    Case Else
        ''''''''''''''''''''''''''''''
        ' Multi-dimensional array.
        ' This is not allowed.
        ''''''''''''''''''''''''''''''
        If NoAlerts = False Then
            MsgBox "SourceArray is a multi-dimensional array. This is not allowed."
        End If
        Exit Function
End Select

'''''''''''''''''''''''''''''''''
' Ensure that all the elements of
' SourceArray are in fact objects.
'''''''''''''''''''''''''''''''''
For InNdx = LBound(SourceArray) To UBound(SourceArray)
    If IsObject(SourceArray(InNdx)) = False Then
        If NoAlerts = False Then
            MsgBox "Element " & CStr(InNdx) & " of SourceArray is not an object."
        End If
        Exit Function
    End If
Next InNdx

''''''''''''''''''''''''''''''
' Erase the ResultArray. Since
' ResultArray is dynamic, this
' will relase the memory used
' by ResultArray and return
' the array to an unallocated
' state.
''''''''''''''''''''''''''''''
Erase ResultArray
''''''''''''''''''''''''''''''
' Now, size ResultArray to the
' size of SourceArray. After
' moving all the non-Nothing
' elements, we'll do another
' resize to get ResultArray
' to the used size. This method
' allows us to avoid Redim
' Preserve for every element.
'''''''''''''''''''''''''''''
ReDim ResultArray(LBound(SourceArray) To UBound(SourceArray))

ResNdx = LBound(SourceArray)
For InNdx = LBound(SourceArray) To UBound(SourceArray)
    If Not SourceArray(InNdx) Is Nothing Then
        Set ResultArray(ResNdx) = SourceArray(InNdx)
        ResNdx = ResNdx + 1
    End If
Next InNdx
''''''''''''''''''''''''''''''''''''''''''
' Now that we've copied all the
' non-Nothing elements from SourceArray
' to ResultArray, we call Redim Preserve
' to resize the ResultArray to the size
' actually used. Test ResNdx to see
' if we actually copied any elements.
''''''''''''''''''''''''''''''''''''''''''
If ResNdx > LBound(SourceArray) Then
    ''''''''''''''''''''''''''''''''''''''''
    ' If ResNdx > LBound(SourceArray) then
    ' we copied at least one element out of
    ' SourceArray.
    ''''''''''''''''''''''''''''''''''''''''
    ReDim Preserve ResultArray(LBound(ResultArray) To ResNdx - 1)
Else
    '''''''''''''''''''''''''''''''''''''''''''''''
    ' Otherwise, we didn't copy any elements
    ' from SourceArray (all elements in SourceArray
    ' were Nothing). In this case, Erase ResultArray.
    '''''''''''''''''''''''''''''''''''''''''''''''''
    Erase ResultArray
End If
'''''''''''''''''''''''''''''
' No errors were encountered.
' Return True.
'''''''''''''''''''''''''''''
CopyNonNothingObjectsToArray = True


End Function



Public Function DataTypeOfArray(Arr As Variant) As VbVarType
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' DataTypeOfArray
'
' Returns a VbVarType value indicating data type of the elements of
' Arr.
'
' The VarType of an array is the value vbArray plus the VbVarType value of the
' data type of the array. For example the VarType of an array of Longs is 8195,
' which equal to vbArray + vbLong. This code subtracts the value of vbArray to
' return the native data type.
'
' If Arr is a simple array, either single- or mulit-
' dimensional, the function returns the data type of the array. Arr
' may be an unallocated array. We can still get the data type of an unallocated
' array.
'
' If Arr is an array of arrays, the function returns vbArray. To retrieve
' the data type of a subarray, pass into the function one of the sub-arrays. E.g.,
' Dim R As VbVarType
' R = DataTypeOfArray(A(LBound(A)))
'
' This function support single and multidimensional arrays. It does not
' support user-defined types. If Arr is an array of empty variants (vbEmpty)
' it returns vbVariant
'
' Returns -1 if Arr is not an array.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim Element As Variant
Dim NumDimensions As Long

' If Arr is not an array, return
' vbEmpty and get out.
If IsArray(Arr) = False Then
    DataTypeOfArray = -1
    Exit Function
End If

If IsArrayEmpty(Arr) = True Then
    ' If the array is unallocated, we can still get its data type.
    ' The result of VarType of an array is vbArray + the VarType
    ' of elements of the array (e.g., the VarType of an array of Longs
    ' is 8195, which is vbArray + vbLong). Thus, to get the basic data
    ' type of the array, we subtract the value vbArray.
    DataTypeOfArray = VarType(Arr) - vbArray
Else
    ' get the number of dimensions in the array.
    NumDimensions = NumberOfArrayDimensions(Arr)
    ' set variable Element to first element of the first dimension
    ' of the array
    If NumDimensions = 1 Then
        If IsObject(Arr(LBound(Arr))) = True Then
            DataTypeOfArray = vbObject
            Exit Function
        End If
        Element = Arr(LBound(Arr))
    Else
        If IsObject(Arr(LBound(Arr), 1)) = True Then
            DataTypeOfArray = vbObject
            Exit Function
        End If
        Element = Arr(LBound(Arr), 1)
    End If
    ' if we were passed an array of arrays, IsArray(Element) will
    ' be true. Therefore, return vbArray. If IsArray(Element) is false,
    ' we weren't passed an array of arrays, so simply return the data type of
    ' Element.
    If IsArray(Element) = True Then
        DataTypeOfArray = vbArray
    Else
        If VarType(Element) = vbEmpty Then
            DataTypeOfArray = vbVariant
        Else
            DataTypeOfArray = VarType(Element)
        End If
    End If
End If


End Function

Public Function DeleteArrayElement(InputArray As Variant, ElementNumber As Long, _
    Optional ResizeDynamic As Boolean = False) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' DeleteArrayElement
' This function deletes an element from InputArray, and shifts elements that are to the
' right of the deleted element to the left. If InputArray is a dynamic array, and the
' ResizeDynamic parameter is True, the array will be resized one element smaller. Otherwise,
' the right-most entry in the array is set to the default value appropriate to the data
' type of the array (0, vbNullString, Empty, or Nothing). If the array is an array of Variant
' types, the default data type is the data type of the last element in the array.
' The function returns True if the elememt was successfully deleted, or False if an error
' occurrred. This procedure works only on single-dimensional
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim Ndx As Long
Dim VType As VbVarType

''''''''''''''''''''''''''''''''
' Set the default result
''''''''''''''''''''''''''''''''
DeleteArrayElement = False

''''''''''''''''''''''''''''''''
' Ensure InputArray is an array.
''''''''''''''''''''''''''''''''
If IsArray(InputArray) = False Then
    Exit Function
End If
```

```
''''''''''''''''''''''''''''''''''''''''''''
' Ensure we have a single dimensional array
''''''''''''''''''''''''''''''''''''''''''''
If NumberOfArrayDimensions(Arr:=InputArray) <> 1 Then
    Exit Function
End If

'''''''''''''''''''''''''''''''''''''''''''''''
' Ensure we have a valid ElementNumber
'''''''''''''''''''''''''''''''''''''''''''''''
If (ElementNumber < LBound(InputArray)) Or (ElementNumber > UBound(InputArray)) Then
    Exit Function
End If

If LBound(InputArray) = UBound(InputArray) Then
    Erase InputArray
    Exit Function
End If
'''''''''''''''''''''''''''''''''''''''''''''''
' Get the variable data type of the element
' we're deleting.
'''''''''''''''''''''''''''''''''''''''''''''''
VType = VarType(InputArray(UBound(InputArray)))
If VType >= vbArray Then
    VType = VType - vbArray
End If
'''''''''''''''''''''''''''''''''''''''''''''''
' Shift everything to the left
'''''''''''''''''''''''''''''''''''''''''''''''
For Ndx = ElementNumber To UBound(InputArray) - 1
    InputArray(Ndx) = InputArray(Ndx + 1)
Next Ndx
'''''''''''''''''''''''''''''''''''''''''''''''
' If ResizeDynamic is True, resize the array
' if it is dynamic.
'''''''''''''''''''''''''''''''''''''''''''''''
If IsArrayDynamic(Arr:=InputArray) = True Then
    If ResizeDynamic = True Then
        ''''''''''''''''''''''''''''''''
        ' Resize the array and get out.
        ''''''''''''''''''''''''''''''''
        ReDim Preserve InputArray(LBound(InputArray) To UBound(InputArray) - 1)
        DeleteArrayElement = True
        Exit Function
    End If
End If
'''''''''''''''''''''''''''''
' Set the last element of the
' InputArray to the proper
' default value.
'''''''''''''''''''''''''''''
Select Case VType
    Case vbByte, vbInteger, vbLong, vbSingle, vbDouble, vbDate, vbCurrency, vbDecimal
        InputArray(UBound(InputArray)) = 0
    Case vbString
        InputArray(UBound(InputArray)) = vbNullString
    Case vbArray, vbVariant, vbEmpty, vbError, vbNull, vbUserDefinedType
        InputArray(UBound(InputArray)) = Empty
    Case vbBoolean
        InputArray(UBound(InputArray)) = False
    Case vbObject
        Set InputArray(UBound(InputArray)) = Nothing
    Case Else
        InputArray(UBound(InputArray)) = 0
End Select

DeleteArrayElement = True

End Function

Public Function FirstNonEmptyStringIndexInArray(InputArray As Variant) As Long
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' FirstNonEmptyStringIndexInArray
' This returns the index into InputArray of the first non-empty string.
' This is generally used when InputArray is the result of a sort operation,
' which puts empty strings at the beginning of the array.
' Returns -1 is an error occurred or if the entire array is empty strings.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim Ndx As Long

If IsArray(InputArray) = False Then
    FirstNonEmptyStringIndexInArray = -1
    Exit Function
End If

Select Case NumberOfArrayDimensions(Arr:=InputArray)
    Case 0
        ''''''''''''''''''''''''''''''''''''''''''
        ' indicates an unallocated dynamic array.
        ''''''''''''''''''''''''''''''''''''''''''
        FirstNonEmptyStringIndexInArray = -1
        Exit Function
    Case 1
        ''''''''''''''''''''''''''''''''''''''''''
        ' single dimensional array. OK.
        ''''''''''''''''''''''''''''''''''''''''''
    Case Else
        ''''''''''''''''''''''''''''''''''''''''''
        ' multidimensional array. Invalid.
        ''''''''''''''''''''''''''''''''''''''''
        FirstNonEmptyStringIndexInArray = -1
        Exit Function
End Select

For Ndx = LBound(InputArray) To UBound(InputArray)
    If InputArray(Ndx) <> vbNullString Then
        FirstNonEmptyStringIndexInArray = Ndx
        Exit Function
    End If
Next Ndx

FirstNonEmptyStringIndexInArray = -1
End Function


Public Function InsertElementIntoArray(InputArray As Variant, Index As Long, _
    Value As Variant) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' InsertElementIntoArray
' This function inserts an element with a value of Value into InputArray at locatation Index.
' InputArray must be a dynamic array. The Value is stored in location Index, and everything
' to the right of Index is shifted to the right. The array is resized to make room for
' the new element. The value of Index must be greater than or equal to the LBound of
' InputArray and less than or equal to UBound+1. If Index is UBound+1, the Value is
' placed at the end of the array.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim Ndx As Long

''''''''''''''''''''''''''''''
' Set the default return value.
''''''''''''''''''''''''''''''
InsertElementIntoArray = False

'''''''''''''''''''''''''''''''
' Ensure InputArray is an array.
'''''''''''''''''''''''''''''''
If IsArray(InputArray) = False Then
    Exit Function
End If

'''''''''''''''''''''''''''''''
' Ensure InputArray is dynamic.
'''''''''''''''''''''''''''''''
If IsArrayDynamic(Arr:=InputArray) = False Then
    Exit Function
End If

''''''''''''''''''''''''''''''''
' Ensure InputArray is allocated.
''''''''''''''''''''''''''''''''
If IsArrayAllocated(Arr:=InputArray) = False Then
    Exit Function
End If

''''''''''''''''''''''''''''''''
' Ensure InputArray is a single
' dimensional array.
''''''''''''''''''''''''''''''''
If NumberOfArrayDimensions(Arr:=InputArray) <> 1 Then
    Exit Function
End If

''''''''''''''''''''''''''''''''''''''''
' Ensure Index is a valid element index.
' We allow Index to be equal to
' UBound + 1 to facilitate inserting
' a value at the end of the array. E.g.,
' InsertElementIntoArray(Arr,UBound(Arr)+1,123)
' will insert 123 at the end of the array.
''''''''''''''''''''''''''''''''''''''''
If (Index < LBound(InputArray)) Or (Index > UBound(InputArray) + 1) Then
    Exit Function
End If

''''''''''''''''''''''''''''''''''''''''''''
' Resize the array
''''''''''''''''''''''''''''''''''''''''''''
ReDim Preserve InputArray(LBound(InputArray) To UBound(InputArray) + 1)
''''''''''''''''''''''''''''''''''''''''''''
' First, we set the newly created last element
' of InputArray to Value. This is done to trap
' an error 13, type mismatch. This last entry
' will be overwritten when we shift elements
' to the right, and the Value will be inserted
' at Index.
''''''''''''''''''''''''''''''''''''''''''''''
On Error Resume Next
Err.Clear
InputArray(UBound(InputArray)) = Value
If Err.Number <> 0 Then
    ''''''''''''''''''''''''''''''''''''''
    ' An error occurred, most likely
    ' an error 13, type mismatch.
    ' Redim the array back to its original
    ' size and exit the function.
    ''''''''''''''''''''''''''''''''''''''
    ReDim Preserve InputArray(LBound(InputArray) To UBound(InputArray) - 1)
    Exit Function
End If
```

```
''''''''''''''''''''''''''''''''''''''''''''''
' Shift everything to the right.
''''''''''''''''''''''''''''''''''''''''''''''
For Ndx = UBound(InputArray) To Index + 1 Step -1
    InputArray(Ndx) = InputArray(Ndx - 1)
Next Ndx

''''''''''''''''''''''''''''''''''''''''''''''
' Insert Value at Index
''''''''''''''''''''''''''''''''''''''''''''''
InputArray(Index) = Value


InsertElementIntoArray = True


End Function



Public Function IsArrayAllDefault(InputArray As Variant) As Boolean
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' IsArrayAllEmpty
' Returns True if the array contains all default values for its
' data type:
'   Variable Type           Value
'   -------------           -------------------
'   Variant                 Empty
'   String                  vbNullString
'   Numeric                 0
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim Ndx As Long
Dim DefaultValue As Variant

'''''''''''''''''''''''''''''''
' Set the default return value.
'''''''''''''''''''''''''''''''
IsArrayAllDefault = False

'''''''''''''''''''''''''''''''''''
' Ensure InputArray is an array.
'''''''''''''''''''''''''''''''''''
If IsArray(InputArray) = False Then
    IsArrayAllDefault = False
    Exit Function
End If

''''''''''''''''''''''''''''''''''
' Ensure array is allocated. An
' unallocated is considered to be
' all the same type. Return True.
''''''''''''''''''''''''''''''''''
If IsArrayAllocated(Arr:=InputArray) = False Then
    IsArrayAllDefault = True
    Exit Function
End If

''''''''''''''''''''''''''''''''''
' Test the type of variable
''''''''''''''''''''''''''''''''''
Select Case VarType(InputArray)
    Case vbArray + vbVariant
        DefaultValue = Empty
    Case vbArray + vbString
        DefaultValue = vbNullString
    Case Is > vbArray
        DefaultValue = 0
End Select
For Ndx = LBound(InputArray) To UBound(InputArray)
    If IsObject(InputArray(Ndx)) Then
        If Not InputArray(Ndx) Is Nothing Then
            Exit Function
        Else

        End If
    Else
        If VarType(InputArray(Ndx)) <> vbEmpty Then
            If InputArray(Ndx) <> DefaultValue Then
                Exit Function
            End If
        End If
    End If
Next Ndx

'''''''''''''''''''''''''''''''
' If we make it out of the loop,
' the array is all defaults.
' Return True.
'''''''''''''''''''''''''''''''
IsArrayAllDefault = True


End Function



Public Function IsArrayAllNumeric(Arr As Variant, _
    Optional AllowNumericStrings As Boolean = False) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' IsArrayAllNumeric
' This function returns True is Arr is entirely numeric. False otherwise. The AllowNumericStrings
' parameter indicates whether strings containing numeric data are considered numeric. If this
' parameter is True, a numeric string is considered a numeric variable. If this parameter is
' omitted or False, a numeric string is not considered a numeric variable.
' Variants that are numeric or Empty are allowed. Variants that are arrays, objects, or
' non-numeric data are not allowed.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim Ndx As Long

''''''''''''''''''''''''''''
' Ensure Arr is an array.
''''''''''''''''''''''''''''
If IsArray(Arr) = False Then
    IsArrayAllNumeric = False
    Exit Function
End If

''''''''''''''''''''''''''''''''''''''
' Ensure Arr is allocated (non-empty).
''''''''''''''''''''''''''''''''''''''
If IsArrayEmpty(Arr:=Arr) = True Then
    IsArrayAllNumeric = False
    Exit Function
End If

''''''''''''''''''''''''''''''''''''''
' Loop through the array.
'''''''''''''''''''''''''''''''''''''
For Ndx = LBound(Arr) To UBound(Arr)
    Select Case VarType(Arr(Ndx))
        Case vbInteger, vbLong, vbDouble, vbSingle, vbCurrency, vbDecimal, vbEmpty
            ' all valid numeric types

        Case vbString
            '''''''''''''''''''''''''''''''''''''''''''''''''''''''
            ' For strings, check the AllowNumericStrings parameter.
            ' If True and the element is a numeric string, allow it.
            ' If it is a non-numeric string, exit with False.
            ' If AllowNumericStrings is False, all strings, even
            ' numeric strings, will cause a result of False.
            '''''''''''''''''''''''''''''''''''''''''''''''''''''''
            If AllowNumericStrings = True Then
                '''''''''''''''''''''''''''''''''
                ' Allow numeric strings.
                '''''''''''''''''''''''''''''''''
                If IsNumeric(Arr(Ndx)) = False Then
                    IsArrayAllNumeric = False
                    Exit Function
                End If
            Else
                IsArrayAllNumeric = False
                Exit Function
            End If
        Case vbVariant
            '''''''''''''''''''''''''''''''''''''''''''''
            ' For Variants, disallow Arrays and Objects.
            ' If the element is not an array or an object,
            ' test whether it is numeric. Allow numeric
            ' Varaints.
            '''''''''''''''''''''''''''''''''''''''''''''
            If IsArray(Arr(Ndx)) = True Then
                IsArrayAllNumeric = False
                Exit Function
            End If
            If IsObject(Arr(Ndx)) = True Then
                IsArrayAllNumeric = False
                Exit Function
            End If

            If IsNumeric(Arr(Ndx)) = False Then
                IsArrayAllNumeric = False
                Exit Function
            End If

        Case Else
            ' any other data type returns False
            IsArrayAllNumeric = False
            Exit Function
    End Select
Next Ndx

IsArrayAllNumeric = True

End Function



Public Function IsArrayAllocated(Arr As Variant) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' IsArrayAllocated
' Returns TRUE if the array is allocated (either a static array or a dynamic array that has been
' sized with Redim) or FALSE if the array is not allocated (a dynamic that has not yet
' been sized with Redim, or a dynamic array that has been Erased). Static arrays are always
' allocated.
'
' The VBA IsArray function indicates whether a variable is an array, but it does not
' distinguish between allocated and unallocated arrays. It will return TRUE for both
' allocated and unallocated arrays. This function tests whether the array has actually
' been allocated.
```

```
'
' This function is just the reverse of IsArrayEmpty.
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim N As Long
On Error Resume Next

' if Arr is not an array, return FALSE and get out.
If IsArray(Arr) = False Then
    IsArrayAllocated = False
    Exit Function
End If

' Attempt to get the UBound of the array. If the array has not been allocated,
' an error will occur. Test Err.Number to see if an error occurred.
N = UBound(Arr, 1)
If (Err.Number = 0) Then
    ''''''''''''''''''''''''''''''''''''''
    ' Under some circumstances, if an array
    ' is not allocated, Err.Number will be
    ' 0. To acccomodate this case, we test
    ' whether LBound <= Ubound. If this
    ' is True, the array is allocated. Otherwise,
    ' the array is not allocated.
    '''''''''''''''''''''''''''''''''''''''
    If LBound(Arr) <= UBound(Arr) Then
        ' no error. array has been allocated.
        IsArrayAllocated = True
    Else
        IsArrayAllocated = False
    End If
Else
    ' error. unallocated array
    IsArrayAllocated = False
End If

End Function



Public Function IsArrayDynamic(ByRef Arr As Variant) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' IsArrayDynamic
' This function returns TRUE or FALSE indicating whether Arr is a dynamic array.
' Note that if you attempt to ReDim a static array in the same procedure in which it is
' declared, you'll get a compiler error and your code won't run at all.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim LUBound As Long

' If we weren't passed an array, get out now with a FALSE result
If IsArray(Arr) = False Then
    IsArrayDynamic = False
    Exit Function
End If

' If the array is empty, it hasn't been allocated yet, so we know
' it must be a dynamic array.
If IsArrayEmpty(Arr:=Arr) = True Then
    IsArrayDynamic = True
    Exit Function
End If

' Save the UBound of Arr.
' This value will be used to restore the original UBound if Arr
' is a single-dimensional dynamic array. Unused if Arr is multi-dimensional,
' or if Arr is a static array.
LUBound = UBound(Arr)

On Error Resume Next
Err.Clear

' Attempt to increase the UBound of Arr and test the value of Err.Number.
' If Arr is a static array, either single- or multi-dimensional, we'll get a
' C_ERR_ARRAY_IS_FIXED_OR_LOCKED error. In this case, return FALSE.
'
' If Arr is a single-dimensional dynamic array, we'll get C_ERR_NO_ERROR error.
'
' If Arr is a multi-dimensional dynamic array, we'll get a
' C_ERR_SUBSCRIPT_OUT_OF_RANGE error.
'
' For either C_NO_ERROR or C_ERR_SUBSCRIPT_OUT_OF_RANGE, return TRUE.
' For C_ERR_ARRAY_IS_FIXED_OR_LOCKED, return FALSE.

ReDim Preserve Arr(LBound(Arr) To LUBound + 1)

Select Case Err.Number
    Case C_ERR_NO_ERROR
        ' We successfully increased the UBound of Arr.
        ' Do a ReDim Preserve to restore the original UBound.
        ReDim Preserve Arr(LBound(Arr) To LUBound)
        IsArrayDynamic = True
    Case C_ERR_SUBSCRIPT_OUT_OF_RANGE
        ' Arr is a multi-dimensional dynamic array.
        ' Return True.
        IsArrayDynamic = True
    Case C_ERR_ARRAY_IS_FIXED_OR_LOCKED
        ' Arr is a static single- or multi-dimensional array.
        ' Return False
        IsArrayDynamic = False
    Case Else
        ' We should never get here.
        ' Some unexpected error occurred. Be safe and return False.
        IsArrayDynamic = False
End Select

End Function


Public Function IsArrayEmpty(Arr As Variant) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' IsArrayEmpty
' This function tests whether the array is empty (unallocated). Returns TRUE or FALSE.
'
' The VBA IsArray function indicates whether a variable is an array, but it does not
' distinguish between allocated and unallocated arrays. It will return TRUE for both
' allocated and unallocated arrays. This function tests whether the array has actually
' been allocated.
'
' This function is really the reverse of IsArrayAllocated.
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim LB As Long
Dim UB As Long

Err.Clear
On Error Resume Next
If IsArray(Arr) = False Then
    ' we weren't passed an array, return True
    IsArrayEmpty = True
End If

' Attempt to get the UBound of the array. If the array is
' unallocated, an error will occur.
UB = UBound(Arr, 1)
If (Err.Number <> 0) Then
    IsArrayEmpty = True
Else
    ''''''''''''''''''''''''''''''''''''''''''
    ' On rare occassion, under circumstances I
    ' cannot reliably replictate, Err.Number
    ' will be 0 for an unallocated, empty array.
    ' On these occassions, LBound is 0 and
    ' UBoung is -1.
    ' To accomodate the weird behavior, test to
    ' see if LB > UB. If so, the array is not
    ' allocated.
    ''''''''''''''''''''''''''''''''''''''''''
    Err.Clear
    LB = LBound(Arr)
    If LB > UB Then
        IsArrayEmpty = True
    Else
        IsArrayEmpty = False
    End If
End If

End Function




Public Function IsArrayObjects(InputArray As Variant, _
    Optional AllowNothing As Boolean = True) As Boolean
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' IsArrayObjects
' Returns True if InputArray is entirely objects (Nothing objects are
' optionally allowed -- default it true, allow Nothing objects). Set the
' AllowNothing to true or false to indicate whether Nothing objects
' are allowed.
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim Ndx As Long

'''''''''''''''''''''''''''''''
' Set the default return value.
'''''''''''''''''''''''''''''''
IsArrayObjects = False

'''''''''''''''''''''''''''''''
' Ensure InputArray is an array.
'''''''''''''''''''''''''''''''
If IsArray(InputArray) = False Then
    Exit Function
End If

''''''''''''''''''''''''''''''''''''
' Ensure we have a single dimensional
' array.
''''''''''''''''''''''''''''''''''''
Select Case NumberOfArrayDimensions(Arr:=InputArray)
    Case 0
        ''''''''''''''''''''''''''''''''''
        ' Unallocated dynamic array.
        ' Not allowed.
        ''''''''''''''''''''''''''''''''''
        Exit Function
    Case 1
```

```
            '''''''''''''''''''''''''''''''''
            ' OK
            '''''''''''''''''''''''''''''''''
        Case Else
            '''''''''''''''''''''''''''''''''
            ' Multi-dimensional array.
            ' Not allowed.
            '''''''''''''''''''''''''''''''''
            Exit Function
    End Select

    For Ndx = LBound(InputArray) To UBound(InputArray)
        If IsObject(InputArray(Ndx)) = False Then
            Exit Function
        End If
        If InputArray(Ndx) Is Nothing Then
            If AllowNothing = False Then
                Exit Function
            End If
        End If
    Next Ndx

    IsArrayObjects = True

    End Function




    Public Function IsNumericDataType(TestVar As Variant) As Boolean
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' IsNumericDataType
    '
    ' This function returns TRUE or FALSE indicating whether the data
    ' type of a variable is a numeric data type. It will return TRUE
    ' for all of the following data types:
    '       vbCurrency
    '       vbDecimal
    '       vbDouble
    '       vbInteger
    '       vbLong
    '       vbSingle
    '
    ' It will return FALSE for any other data type, including empty Variants and objects.
    ' If TestVar is an allocated array, it will test data type of the array
    ' and return TRUE or FALSE for that data type. If TestVar is an allocated
    ' array, it tests the data type of the first element of the array. If
    ' TestVar is an array of Variants, the function will indicate only whether
    ' the first element of the array is numeric. Other elements of the array
    ' may not be numeric data types. To test an entire array of variants
    ' to ensure they are all numeric data types, use the IsVariantArrayNumeric
    ' function.
    '
    ' It will return FALSE for any other data type. Use this procedure
    ' instead of VBA's IsNumeric function because IsNumeric will return
    ' TRUE if the variable is a string containing numeric data. This
    ' will cause problems with code like
    '        Dim V1 As Variant
    '        Dim V2 As Variant
    '        V1 = "1"
    '        V2 = "2"
    '        If IsNumeric(V1) = True Then
    '            If IsNumeric(V2) = True Then
    '                Debug.Print V1 + V2
    '            End If
    '        End If
    '
    ' The output of the Debug.Print statement will be "12", not 3,
    ' because V1 and V2 are strings and the '+' operator acts like
    ' the '&' operator when used with strings. This can lead to
    ' unexpected results.
    '
    ' IsNumeric should only be used to test strings for numeric content
    ' when converting a string value to a numeric variable.
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
        Dim Element As Variant
        Dim NumDims As Long
        If IsArray(TestVar) = True Then
            NumDims = NumberOfArrayDimensions(Arr:=TestVar)
            If NumDims > 1 Then
                '''''''''''''''''''''''''''''''''
                ' this procedure does not support
                ' multi-dimensional arrays.
                '''''''''''''''''''''''''''''''''
                IsNumericDataType = False
                Exit Function
            End If
            If IsArrayAllocated(Arr:=TestVar) = True Then
                Element = TestVar(LBound(TestVar))
                Select Case VarType(Element)
                    Case vbCurrency, vbDecimal, vbDouble, vbInteger, vbLong, vbSingle
                        IsNumericDataType = True
                        Exit Function
                    Case Else
                        IsNumericDataType = False
                        Exit Function
                End Select
            Else
                Select Case VarType(TestVar) - vbArray
                    Case vbCurrency, vbDecimal, vbDouble, vbInteger, vbLong, vbSingle
                        IsNumericDataType = True
                        Exit Function
                    Case Else
                        IsNumericDataType = False
                        Exit Function
                End Select
            End If
        End If
        Select Case VarType(TestVar)
            Case vbCurrency, vbDecimal, vbDouble, vbInteger, vbLong, vbSingle
                IsNumericDataType = True
            Case Else
                IsNumericDataType = False
        End Select
    End Function



    Public Function IsVariantArrayConsistent(Arr As Variant) As Boolean
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' IsVariantArrayConsistent
    '
    ' This returns TRUE or FALSE indicating whether an array of variants
    ' contains all the same data types. Returns FALSE under the following
    ' circumstances:
    '       Arr is not an array
    '       Arr is an array but is unallocated
    '       Arr is a multidimensional array
    '       Arr is allocated but does not contain consistant data types.
    '
    ' If Arr is an array of objects, objects that are Nothing are ignored.
    ' As long as all non-Nothing objects are the same object type, the
    ' function returns True.
    '
    ' It returns TRUE if all the elements of the array have the same
    ' data type. If Arr is an array of a specific data types, not variants,
    ' (E.g., Dim V(1 To 3) As Long), the function will return True. If
    ' an array of variants contains an uninitialized element (VarType =
    ' vbEmpty) that element is skipped and not used in the comparison. The
    ' reasoning behind this is that an empty variable will return the
    ' data type of the variable to which it is assigned (e.g., it will
    ' return vbNullString to a String and 0 to a Double).
    '
    ' The function does not support arrays of User Defined Types.
    '
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''


    Dim FirstDataType As VbVarType
    Dim Ndx As Long
    ''''''''''''''''''''''''''''''''''''''''''
    ' Exit with False if Arr is not an array.
    ''''''''''''''''''''''''''''''''''''''''''
    If IsArray(Arr) = False Then
        IsVariantArrayConsistent = False
        Exit Function
    End If

    '''''''''''''''''''''''''''''''''''''''''''
    ' Exit with False if Arr is not allocated.
    '''''''''''''''''''''''''''''''''''''''''''
    If IsArrayAllocated(Arr) = False Then
        IsVariantArrayConsistent = False
        Exit Function
    End If
    '''''''''''''''''''''''''''''''''''''''''''
    ' Exit with false on multi-dimensional
    ' arrays.
    '''''''''''''''''''''''''''''''''''''''''''
    If NumberOfArrayDimensions(Arr) <> 1 Then
        IsVariantArrayConsistent = False
        Exit Function
    End If

    '''''''''''''''''''''''''''''''''''''''''''
    ' Test if we have an array of a specific
    ' type rather than Variants. If so,
    ' return TRUE and get out.
    '''''''''''''''''''''''''''''''''''''''''''
    If (VarType(Arr) <= vbArray) And _
        (VarType(Arr) <> vbVariant) Then
        IsVariantArrayConsistent = True
        Exit Function
    End If

    '''''''''''''''''''''''''''''''''''''''''''
    ' Get the data type of the first element.
    '''''''''''''''''''''''''''''''''''''''''''
    FirstDataType = VarType(Arr(LBound(Arr)))
    '''''''''''''''''''''''''''''''''''''''''''
    ' Loop through the array and exit if
    ' a differing data type if found.
    '''''''''''''''''''''''''''''''''''''''''''
    For Ndx = LBound(Arr) + 1 To UBound(Arr)
        If VarType(Arr(Ndx)) <> vbEmpty Then
            If IsObject(Arr(Ndx)) = True Then
                If Not Arr(Ndx) Is Nothing Then
                    If VarType(Arr(Ndx)) <> FirstDataType Then
                        IsVariantArrayConsistent = False
                        Exit Function
                    End If
                End If
            Else
                If VarType(Arr(Ndx)) <> FirstDataType Then
                    IsVariantArrayConsistent = False
                    Exit Function
                End If
            End If
        End If
    Next Ndx

    '''''''''''''''''''''''''''''''''''''''''
    ' If we make it out of the loop,
    ' then the array is consistent.
    '''''''''''''''''''''''''''''''''''''''''
    IsVariantArrayConsistent = True

    End Function



    Public Function IsVariantArrayNumeric(TestArray As Variant) As Boolean
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' IsVariantArrayNumeric
    '
    ' This function return TRUE if all the elements of an array of
    ' variants are numeric data types. They need not all be the same data
    ' type. You can have a mix of Integer, Longs, Doubles, and Singles.
    ' As long as they are all numeric data types, the function will
    ' return TRUE. If a non-numeric data type is encountered, the
    ' function will return FALSE. Also, it will return FALSE if
    ' TestArray is not an array, or if TestArray has not been
    ' allocated. TestArray may be a multi-dimensional array. This
    ' procedure uses the IsNumericDataType function to determine whether
    ' a variable is a numeric data type. If there is an uninitialized
    ' variant (VarType = vbEmpty) in the array, it is skipped and not
    ' used in the comparison (i.e., Empty is considered a valid numeric
    ' data type since you can assign a number to it).
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

    Dim Ndx As Long
    Dim DimNdx As Long
    Dim NumDims As Long

    ''''''''''''''''''''''''''''''''
    ' Ensure TestArray is an array.
    ''''''''''''''''''''''''''''''''
    If IsArray(TestArray) = False Then
        IsVariantArrayNumeric = False
        Exit Function
    End If
    ''''''''''''''''''''''''''''''''
    ' Ensure that TestArray has been
    ' allocated.
    ''''''''''''''''''''''''''''''''
    If IsArrayAllocated(Arr:=TestArray) = False Then
        IsVariantArrayNumeric = False
        Exit Function
    End If
    ''''''''''''''''''''''''''''''''''''
    ' Ensure the array is a one
    ' dimensional array. This procedure
    ' will not work on multi-dimensional
    ' arrays.
    ''''''''''''''''''''''''''''''''''''
    'If NumberOfArrayDimensions(Arr:=TestArray) > 1 Then
    '    IsVariantArrayNumeric = False
    '    Exit Function
    'End If

    NumDims = NumberOfArrayDimensions(Arr:=TestArray)
    If NumDims = 1 Then
        '''''''''''''''''''''''''''''''''''
        ' single dimensional array
        '''''''''''''''''''''''''''''''''''
        For Ndx = LBound(TestArray) To UBound(TestArray)
            If IsObject(TestArray(Ndx)) = True Then
                IsVariantArrayNumeric = False
                Exit Function
            End If

            If VarType(TestArray(Ndx)) <> vbEmpty Then
                If IsNumericDataType(TestVar:=TestArray(Ndx)) = False Then
                    IsVariantArrayNumeric = False
                    Exit Function
                End If
            End If
        Next Ndx
    Else
        ''''''''''''''''''''''''''''''''''''
        ' multi-dimensional array
        ''''''''''''''''''''''''''''''''''''
        For DimNdx = 1 To NumDims
            For Ndx = LBound(TestArray, DimNdx) To UBound(TestArray, DimNdx)
                If VarType(TestArray(Ndx, DimNdx)) <> vbEmpty Then
                    If IsNumericDataType(TestVar:=TestArray(Ndx, DimNdx)) = False Then
                        IsVariantArrayNumeric = False
                        Exit Function
                    End If
                End If
            Next Ndx
        Next DimNdx
    End If

    '''''''''''''''''''''''''''''''''''''''
    ' If we made it out of the loop, then
    ' the array is entirely numeric.
    '''''''''''''''''''''''''''''''''''''''
    IsVariantArrayNumeric = True

    End Function



    Public Function MoveEmptyStringsToEndOfArray(InputArray As Variant) As Boolean
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' This procedure takes the SORTED array InputArray, which, if sorted in
    ' ascending order, will have all empty strings at the front of the array.
    ' This procedure moves those strings to the end of the array, shifting
    ' the non-empty strings forward in the array.
    ' Note that InputArray MUST be sorted in ascending order.
    ' Returns True if the array was correctly shifted (if necessary) and False
    ' if an error occurred.
    ' This function uses the following functions, which are included as Private
    ' procedures at the end of this module.
    '       FirstNonEmptyStringIndexInArray
    '       NumberOfArrayDimensions
    '       IsArrayAllocated
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    Dim Temp As String
    Dim Ndx As Long
    Dim Ndx2 As Long
    Dim NonEmptyNdx As Long
    Dim FirstNonEmptyNdx As Long


    ''''''''''''''''''''''''''''''''
    ' Ensure InpuyArray is an array.
    ''''''''''''''''''''''''''''''''
    If IsArray(InputArray) = False Then
        MoveEmptyStringsToEndOfArray = False
        Exit Function
    End If


    ''''''''''''''''''''''''''''''''''''
    ' Ensure that the array is allocated
    ' (not an empty array).
    ''''''''''''''''''''''''''''''''''''
    If IsArrayAllocated(Arr:=InputArray) = False Then
        MoveEmptyStringsToEndOfArray = False
        Exit Function
    End If


    FirstNonEmptyNdx = FirstNonEmptyStringIndexInArray(InputArray:=InputArray)
    If FirstNonEmptyNdx <= LBound(InputArray) Then
        ''''''''''''''''''''''''''''''''''''''''''
        ' No empty strings at the beginning of the
        ' array. Get out now.
        ''''''''''''''''''''''''''''''''''''''''''
        MoveEmptyStringsToEndOfArray = True
        Exit Function
    End If


    ''''''''''''''''''''''''''''''''''''''''''''''''
    ' Loop through the array, swapping vbNullStrings
    ' at the beginning with values at the end.
    ''''''''''''''''''''''''''''''''''''''''''''''''
    NonEmptyNdx = FirstNonEmptyNdx
    For Ndx = LBound(InputArray) To UBound(InputArray)
        If InputArray(Ndx) = vbNullString Then
            InputArray(Ndx) = InputArray(NonEmptyNdx)
            InputArray(NonEmptyNdx) = vbNullString
            NonEmptyNdx = NonEmptyNdx + 1
            If NonEmptyNdx > UBound(InputArray) Then
                Exit For
            End If
        End If
    Next Ndx
    ''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Set entires (Ndx+1) to UBound(InputArray) to
    ' vbNullStrings.
    ''''''''''''''''''''''''''''''''''''''''''''''''''''
    For Ndx2 = Ndx + 1 To UBound(InputArray)
        InputArray(Ndx2) = vbNullString
    Next Ndx2
    MoveEmptyStringsToEndOfArray = True

    End Function



    Public Function NumberOfArrayDimensions(Arr As Variant) As Integer
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' NumberOfArrayDimensions
    ' This function returns the number of dimensions of an array. An unallocated dynamic array
    ' has 0 dimensions. This condition can also be tested with IsArrayEmpty.
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
Dim Ndx As Integer
Dim Res As Integer
On Error Resume Next
' Loop, increasing the dimension index Ndx, until an error occurs.
' An error will occur when Ndx exceeds the number of dimension
' in the array. Return Ndx - 1.
Do
    Ndx = Ndx + 1
    Res = UBound(Arr, Ndx)
Loop Until Err.Number <> 0

NumberOfArrayDimensions = Ndx - 1

End Function



Public Function NumElements(Arr As Variant, Optional Dimension = 1) As Long
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' NumElements
' Returns the number of elements in the specified dimension (Dimension) of the array in
' Arr. If you omit Dimension, the first dimension is used. The function will return
' 0 under the following circumstances:
'     Arr is not an array, or
'     Arr is an unallocated array, or
'     Dimension is greater than the number of dimension of Arr, or
'     Dimension is less than 1.
'
' This function does not support arrays of user-defined Type variables.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim NumDimensions As Long

' if Arr is not an array, return 0 and get out.
If IsArray(Arr) = False Then
    NumElements = 0
    Exit Function
End If

' if the array is unallocated, return 0 and get out.
If IsArrayEmpty(Arr) = True Then
    NumElements = 0
    Exit Function
End If

' ensure that Dimension is at least 1.
If Dimension < 1 Then
    NumElements = 0
    Exit Function
End If

' get the number of dimensions
NumDimensions = NumberOfArrayDimensions(Arr)
If NumDimensions < Dimension Then
    NumElements = 0
    Exit Function
End If

' returns the number of elements in the array
NumElements = UBound(Arr, Dimension) - LBound(Arr, Dimension) + 1

End Function

Public Function ResetVariantArrayToDefaults(InputArray As Variant) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' ResetVariantArrayToDefaults
' This resets all the elements of an array of Variants back to their appropriate
' default values. The elements of the array may be of mixed types (e.g., some Longs,
' some Objects, some Strings, etc). Each data type will be set to the appropriate
' default value (0, vbNullString, Empty, or Nothing). It returns True if the
' array was set to defautls, or False if an error occurred. InputArray must be
' an allocated single-dimensional array. This function differs from the Erase
' function in that it preserves the original data types, while Erase sets every
' element to Empty.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim Ndx As Long
'''''''''''''''''''''''''''''''
' Set the default return value.
'''''''''''''''''''''''''''''''
ResetVariantArrayToDefaults = False

'''''''''''''''''''''''''''''''
' Ensure InputArray is an array
'''''''''''''''''''''''''''''''
If IsArray(InputArray) = False Then
    Exit Function
End If

'''''''''''''''''''''''''''''''
' Ensure InputArray is a single
' dimensional allocated array.
'''''''''''''''''''''''''''''''
If NumberOfArrayDimensions(Arr:=InputArray) <> 1 Then
    Exit Function
End If

For Ndx = LBound(InputArray) To UBound(InputArray)
    SetVariableToDefault InputArray(Ndx)
Next Ndx
```

```
ResetVariantArrayToDefaults = True

End Function



Public Function ReverseArrayInPlace(InputArray As Variant, _
    Optional NoAlerts As Boolean = False) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' ReverseArrayInPlace
' This procedure reverses the order of an array in place -- this is, the array variable
' in the calling procedure is reversed. This works only on single-dimensional arrays
' of simple data types (String, Single, Double, Integer, Long). It will not work
' on arrays of objects. Use ReverseArrayOfObjectsInPlace to reverse an array of objects.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim Temp As Variant
Dim Ndx As Long
Dim Ndx2 As Long


'''''''''''''''''''''''''''''''''
' Set the default return value.
'''''''''''''''''''''''''''''''''
ReverseArrayInPlace = False

'''''''''''''''''''''''''''''''''
' ensure we have an array
'''''''''''''''''''''''''''''''''
If IsArray(InputArray) = False Then
    If NoAlerts = False Then
        MsgBox "The InputArray parameter is not an array."
    End If
    Exit Function
End If

''''''''''''''''''''''''''''''''''''''
' Test the number of dimensions of the
' InputArray. If 0, we have an empty,
' unallocated array. Get out with
' an error message. If greater than
' one, we have a multi-dimensional
' array, which is not allowed. Only
' an allocated 1-dimensional array is
' allowed.
''''''''''''''''''''''''''''''''''''''
Select Case NumberOfArrayDimensions(InputArray)
    Case 0
        If NoAlerts = False Then
            MsgBox "The input array is an empty, unallocated array."
        End If
        Exit Function
    Case 1
        ' ok
    Case Else
        If NoAlerts = False Then
            MsgBox "The input array is multi-dimensional. ReverseArrayInPlace works only " & _
                   "on single-dimensional arrays."
        End If
        Exit Function
End Select

Ndx2 = UBound(InputArray)
''''''''''''''''''''''''''''''''''''''
' loop from the LBound of InputArray to
' the midpoint of InputArray
''''''''''''''''''''''''''''''''''''''
For Ndx = LBound(InputArray) To ((UBound(InputArray) - LBound(InputArray) + 1) \ 2)
    'swap the elements
    Temp = InputArray(Ndx)
    InputArray(Ndx) = InputArray(Ndx2)
    InputArray(Ndx2) = Temp
    ' decrement the upper index
    Ndx2 = Ndx2 - 1
Next Ndx

''''''''''''''''''''''''''''''''''''''
' OK - Return True
''''''''''''''''''''''''''''''''''''''
ReverseArrayInPlace = True

End Function



Public Function ReverseArrayOfObjectsInPlace(InputArray As Variant, _
    Optional NoAlerts As Boolean = False) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' ReverseArrayOfObjectsInPlace
' This procedure reverses the order of an array in place -- this is, the array variable
' in the calling procedure is reversed. This works only with arrays of objects. It does
' not work on simple variables. Use ReverseArrayInPlace for simple variables. An error
' will occur if an element of the array is not an object.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim Temp As Variant
Dim Ndx As Long
Dim Ndx2 As Long
```

```
''''''''''''''''''''''''''''''''
' Set the default return value.
''''''''''''''''''''''''''''''''
ReverseArrayOfObjectsInPlace = False

''''''''''''''''''''''''''''''''
' ensure we have an array
''''''''''''''''''''''''''''''''
If IsArray(InputArray) = False Then
    If NoAlerts = False Then
        MsgBox "The InputArray parameter is not an array."
    End If
    Exit Function
End If

''''''''''''''''''''''''''''''''''''''
' Test the number of dimensions of the
' InputArray. If 0, we have an empty,
' unallocated array. Get out with
' an error message. If greater than
' one, we have a multi-dimensional
' array, which is not allowed. Only
' an allocated 1-dimensional array is
' allowed.
''''''''''''''''''''''''''''''''''''''
Select Case NumberOfArrayDimensions(InputArray)
    Case 0
        If NoAlerts = False Then
            MsgBox "The input array is an empty, unallocated array."
        End If
        Exit Function
    Case 1
        ' ok
    Case Else
        If NoAlerts = False Then
            MsgBox "The input array is multi-dimensional. ReverseArrayInPlace works only " & _
                   "on single-dimensional arrays."
        End If
        Exit Function
End Select

Ndx2 = UBound(InputArray)

'''''''''''''''''''''''''''''''''''''
' ensure the entire array consists
' of objects (Nothing objects are
' allowed).
'''''''''''''''''''''''''''''''''''''
For Ndx = LBound(InputArray) To UBound(InputArray)
    If IsObject(InputArray(Ndx)) = False Then
        If NoAlerts = False Then
            MsgBox "Array item " & CStr(Ndx) & " is not an object."
        End If
        Exit Function
    End If
Next Ndx

''''''''''''''''''''''''''''''''''''''
' loop from the LBound of InputArray to
' the midpoint of InputArray
''''''''''''''''''''''''''''''''''''''
For Ndx = LBound(InputArray) To ((UBound(InputArray) - LBound(InputArray) + 1) \ 2)
    Set Temp = InputArray(Ndx)
    Set InputArray(Ndx) = InputArray(Ndx2)
    Set InputArray(Ndx2) = Temp
    ' decrement the upper index
    Ndx2 = Ndx2 - 1
Next Ndx

''''''''''''''''''''''''''''''''''''''
' OK - Return True
''''''''''''''''''''''''''''''''''''''
ReverseArrayOfObjectsInPlace = True

End Function


Public Function SetObjectArrayToNothing(InputArray As Variant) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' SetObjectArrrayToNothing
' This sets all the elements of InputArray to Nothing. Use this function
' rather than Erase because if InputArray is an array of Variants, Erase
' will set each element to Empty, not Nothing, and the element will cease
' to be an object.
'
' The function returns True if successful, False otherwise.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Dim N As Long

''''''''''''''''''''''''''''''''''''''
' Ensure InputArray is an array.
''''''''''''''''''''''''''''''''''''''
If IsArray(InputArray) = False Then
    SetObjectArrayToNothing = False
    Exit Function
End If
```

```
''''''''''''''''''''''''''''''''''''''''''''
' Ensure we have a single-dimensional array.
''''''''''''''''''''''''''''''''''''''''''''
If NumberOfArrayDimensions(Arr:=InputArray) <> 1 Then
    SetObjectArrayToNothing = False
    Exit Function
End If

''''''''''''''''''''''''''''''''''''''''''''''''
' Ensure the array is allocated and that each
' element is an object (or Nothing). If the
' array is not allocated, return True.
' We do this test before setting any element
' to Nothing so we don't end up with an array
' that is a mix of Empty and Nothing values.
' This means looping through the array twice,
' but it ensures all or none of the elements
' get set to Nothing.
''''''''''''''''''''''''''''''''''''''''''''''''
If IsArrayAllocated(Arr:=InputArray) = True Then
    For N = LBound(InputArray) To UBound(InputArray)
        If IsObject(InputArray(N)) = False Then
            SetObjectArrayToNothing = False
            Exit Function
        End If
    Next N
Else
    SetObjectArrayToNothing = True
    Exit Function
End If


'''''''''''''''''''''''''''''''''''''''''''''
' Set each element of InputArray to Nothing.
'''''''''''''''''''''''''''''''''''''''''''''
For N = LBound(InputArray) To UBound(InputArray)
    Set InputArray(N) = Nothing
Next N

SetObjectArrayToNothing = True

End Function

Public Function AreDataTypesCompatible(DestVar As Variant, SourceVar As Variant) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' AreDataTypesCompatible
' This function determines if SourceVar is compatiable with DestVar. If the two
' data types are the same, they are compatible. If the value of SourceVar can
' be stored in DestVar with no loss of precision or an overflow, they are compatible.
' For example, if DestVar is a Long and SourceVar is an Integer, they are compatible
' because an integer can be stored in a Long with no loss of information. If DestVar
' is a Long and SourceVar is a Double, they are not compatible because information
' will be lost converting from a Double to a Long (the decimal portion will be lost).
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim SVType As VbVarType
Dim DVType As VbVarType

'''''''''''''''''''''''''''''''''''
' Set the default return type.
'''''''''''''''''''''''''''''''''''
AreDataTypesCompatible = False

'''''''''''''''''''''''''''''''''''
' If DestVar is an array, get the
' type of array. If it is an array
' its VarType is vbArray + VarType(element)
' so we subtract vbArray to get then
' data type of the aray. E.g.,
' the VarType of an array of Longs
' is 8195 = vbArray + vbLong,
' 8195 - vbArray = vbLong (=3).
'''''''''''''''''''''''''''''''''''
If IsArray(DestVar) = True Then
    DVType = VarType(DestVar) - vbArray
Else
    DVType = VarType(DestVar)
End If
'''''''''''''''''''''''''''''''''''
' If SourceVar is an array, get the
' type of array.
'''''''''''''''''''''''''''''''''''
If IsArray(SourceVar) = True Then
    SVType = VarType(SourceVar) - vbArray
Else
    SVType = VarType(SourceVar)
End If

''''''''''''''''''''''''''''''''''''
' If one variable is an array and
' the other is not an array, they
' are incompatible.
''''''''''''''''''''''''''''''''''''
If ((IsArray(DestVar) = True) And (IsArray(SourceVar) = False) Or _
    (IsArray(DestVar) = False) And (IsArray(SourceVar) = True)) Then
    Exit Function
End If


''''''''''''''''''''''''''''''''''''
' Test the data type of DestVar
```

```
    ' and return a result if SourceVar
    ' is compatible with that type.
    ''''''''''''''''''''''''''''''''''''
    If SVType = DVType Then
        '''''''''''''''''''''''''''''''''
        ' The the variable types are the
        ' same, they are compatible.
        ''''''''''''''''''''''''''''''''
        AreDataTypesCompatible = True
        Exit Function
    Else
        '''''''''''''''''''''''''''''''''''''''''
        ' If the data types are not the same,
        ' determine whether they are compatible.
        '''''''''''''''''''''''''''''''''''''''''
        Select Case DVType
            Case vbInteger
                Select Case SVType
                    Case vbInteger
                        AreDataTypesCompatible = True
                        Exit Function
                    Case Else
                        AreDataTypesCompatible = False
                        Exit Function
                End Select

            Case vbLong
                Select Case SVType
                    Case vbInteger, vbLong
                        AreDataTypesCompatible = True
                        Exit Function
                    Case Else
                        AreDataTypesCompatible = False
                        Exit Function
                End Select
            Case vbSingle
                Select Case SVType
                    Case vbInteger, vbLong, vbSingle
                        AreDataTypesCompatible = True
                        Exit Function
                    Case Else
                        AreDataTypesCompatible = False
                        Exit Function
                End Select
            Case vbDouble
                Select Case SVType
                    Case vbInteger, vbLong, vbSingle, vbDouble
                        AreDataTypesCompatible = True
                        Exit Function
                    Case Else
                        AreDataTypesCompatible = False
                        Exit Function
                End Select
            Case vbString
                Select Case SVType
                    Case vbString
                        AreDataTypesCompatible = True
                        Exit Function
                    Case Else
                        AreDataTypesCompatible = False
                        Exit Function
                End Select
            Case vbObject
                Select Case SVType
                    Case vbObject
                        AreDataTypesCompatible = True
                        Exit Function
                    Case Else
                        AreDataTypesCompatible = False
                        Exit Function
                End Select
            Case vbBoolean
                Select Case SVType
                    Case vbBoolean, vbInteger
                        AreDataTypesCompatible = True
                        Exit Function
                    Case Else
                        AreDataTypesCompatible = False
                        Exit Function
                End Select
             Case vbByte
                Select Case SVType
                    Case vbByte
                        AreDataTypesCompatible = True
                        Exit Function
                    Case Else
                        AreDataTypesCompatible = False
                        Exit Function
                End Select
            Case vbCurrency
                Select Case SVType
                    Case vbInteger, vbLong, vbSingle, vbDouble
                        AreDataTypesCompatible = True
                        Exit Function
                    Case Else
                        AreDataTypesCompatible = False
                        Exit Function
                End Select
            Case vbDecimal
                Select Case SVType
                    Case vbInteger, vbLong, vbSingle, vbDouble
                        AreDataTypesCompatible = True
                        Exit Function
                    Case Else
                        AreDataTypesCompatible = False
                        Exit Function
                End Select
            Case vbDate
                Select Case SVType
                    Case vbLong, vbSingle, vbDouble
                        AreDataTypesCompatible = True
                        Exit Function
                    Case Else
                        AreDataTypesCompatible = False
                        Exit Function
                End Select

             Case vbEmpty
                Select Case SVType
                    Case vbVariant
                        AreDataTypesCompatible = True
                        Exit Function
                    Case Else
                        AreDataTypesCompatible = False
                        Exit Function
                End Select
             Case vbError
                AreDataTypesCompatible = False
                Exit Function
             Case vbNull
                AreDataTypesCompatible = False
                Exit Function
             Case vbObject
                Select Case SVType
                    Case vbObject
                        AreDataTypesCompatible = True
                        Exit Function
                    Case Else
                        AreDataTypesCompatible = False
                        Exit Function
                End Select
             Case vbVariant
                AreDataTypesCompatible = True
                Exit Function

        End Select
    End If


    End Function

    Public Sub SetVariableToDefault(ByRef Variable As Variant)
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' SetVariableToDefault
    ' This procedure sets Variable to the appropriate default
    ' value for its data type. Note that it cannot change User-Defined
    ' Types.
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    If IsObject(Variable) Then
        '''''''''''''''''''''''''''''''''''''''''''
        ' We test with IsObject here so that
        ' the object itself, not the default
        ' property of the object, is evaluated.
        ''''''''''''''''''''''''''''''''''''''''''''
        Set Variable = Nothing
    Else
        Select Case VarType(Variable)
            Case Is >= vbArray
                ''''''''''''''''''''''''''''''''''''''''''''
                ' The VarType of an array is
                ' equal to vbArray + VarType(ArrayElement).
                ' Here we check for anything >= vbArray
                ''''''''''''''''''''''''''''''''''''''''''''
                Erase Variable
            Case vbBoolean
                Variable = False
            Case vbByte
                Variable = CByte(0)
            Case vbCurrency
                Variable = CCur(0)
            Case vbDataObject
                Set Variable = Nothing
            Case vbDate
                Variable = CDate(0)
            Case vbDecimal
                Variable = CDec(0)
            Case vbDouble
                Variable = CDbl(0)
            Case vbEmpty
                Variable = Empty
            Case vbError
                Variable = Empty
            Case vbInteger
                Variable = CInt(0)
            Case vbLong
                Variable = CLng(0)
            Case vbNull
                Variable = Empty
            Case vbObject
                Set Variable = Nothing
            Case vbSingle
                Variable = CSng(0)
            Case vbString
                Variable = vbNullString
            Case vbUserDefinedType
                '''''''''''''''''''''''''''''''''
                ' User-Defined-Types cannot be
                ' set to a general default value.
                ' Each element must be explicitly
                ' set to its default value. No
                ' assignment takes place in this
                ' procedure.
                ''''''''''''''''''''''''''''''''''
            Case vbVariant
                ''''''''''''''''''''''''''''''''''''''''''''''''
                ' This case is included for constistancy,
                ' but we will never get here. If the Variant
                ' contains data, VarType returns the type of
                ' that data. An Empty Variant is type vbEmpty.
                ''''''''''''''''''''''''''''''''''''''''''''''''
                Variable = Empty
        End Select
    End If

    End Sub

    Public Function TransposeArray(InputArr As Variant, OutputArr As Variant) As Boolean
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' TransposeArray
    ' This transposes a two-dimensional array. It returns True if successful or
    ' False if an error occurs. InputArr must be two-dimensions. OutputArr must be
    ' a dynamic array. It will be Erased and resized, so any existing content will
    ' be destroyed.
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

    Dim RowNdx As Long
    Dim ColNdx As Long
    Dim LB1 As Long
    Dim LB2 As Long
    Dim UB1 As Long
    Dim UB2 As Long

    '''''''''''''''''''''''''''''''''''
    ' Ensure InputArr and OutputArr
    ' are arrays.
    '''''''''''''''''''''''''''''''''''
    If (IsArray(InputArr) = False) Or (IsArray(OutputArr) = False) Then
        TransposeArray = False
        Exit Function
    End If

    '''''''''''''''''''''''''''''''''''
    ' Ensure OutputArr is a dynamic
    ' array.
    '''''''''''''''''''''''''''''''''''
    If IsArrayDynamic(Arr:=OutputArr) = False Then
        TransposeArray = False
        Exit Function
    End If

    ''''''''''''''''''''''''''''''''''
    ' Ensure InputArr is two-dimensions,
    ' no more, no lesss.
    ''''''''''''''''''''''''''''''''''
    If NumberOfArrayDimensions(Arr:=InputArr) <> 2 Then
        TransposeArray = False
        Exit Function
    End If

    '''''''''''''''''''''''''''''''''''''''
    ' Get the Lower and Upper bounds of
    ' InputArr.
    '''''''''''''''''''''''''''''''''''''''
    LB1 = LBound(InputArr, 1)
    LB2 = LBound(InputArr, 2)
    UB1 = UBound(InputArr, 1)
    UB2 = UBound(InputArr, 2)

    '''''''''''''''''''''''''''''''''''''''''
    ' Erase and ReDim OutputArr
    '''''''''''''''''''''''''''''''''''''''''
    Erase OutputArr
    ReDim OutputArr(LB2 To LB2 + UB2 - LB2, LB1 To LB1 + UB1 - LB1)

    For RowNdx = LBound(InputArr, 2) To UBound(InputArr, 2)
        For ColNdx = LBound(InputArr, 1) To UBound(InputArr, 1)
            OutputArr(RowNdx, ColNdx) = InputArr(ColNdx, RowNdx)
        Next ColNdx
    Next RowNdx

    TransposeArray = True

    End Function

    Public Function VectorsToArray(Arr As Variant, ParamArray Vectors()) As Boolean
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' VectorsToArray
    ' This function takes 1 or more single-dimensional arrays and converts
    ' them into a single multi-dimensional array. Each array in Vectors
    ' comprises one row of the new array. The number of columns in the
    ' new array is the maximum of the number of elements in each vector.
    ' Arr MUST be a dynamic array of a data type compatible with ALL the
    ' elements in each Vector. The code does NOT trap for an error
    ' 13 - Type Mismatch.
    '
    ' If the Vectors are of differing sizes, Arr is sized to hold the
    ' maximum number of elements in a Vector. The procedure Erases the
    ' Arr array, so when it is reallocated with Redim, all elements will
    ' be the reset to their default value (0 or vbNullString or Empty).
    ' Unused elements in the new array will remain the default value for
    ' that data type.
    '
    ' Each Vector in Vectors must be a single dimensional array, but
    ' the Vectors may be of different sizes and LBounds.
    '
    ' Each element in each Vector must be a simple data type. The elements
    ' may NOT be Object, Arrays, or User-Defined Types.
    '
    ' The rows and columns of the result array are 0-based, regardless of
    ' the LBound of each vector and regardless of the Option Base statement.
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    Dim Vector As Variant
    Dim VectorNdx As Long
    Dim NumElements As Long
    Dim NumRows As Long
    Dim NumCols As Long
    Dim RowNdx As Long
    Dim ColNdx As Long
    Dim VType As VbVarType

    ''''''''''''''''''''''''''''''''''
    ' Ensure we have an Array
    '''''''''''''''''''''''''''''''''
    If IsArray(Arr) = False Then
        VectorsToArray = False
        Exit Function
    End If

    '''''''''''''''''''''''''''''''''
    ' Ensure we have a dynamic array
    '''''''''''''''''''''''''''''''''
    If IsArrayDynamic(Arr) = False Then
        VectorsToArray = False
        Exit Function
    End If

    ''''''''''''''''''''''''''''''''
    ' Ensure that at least one vector
    ' was passed in Vectors
    ''''''''''''''''''''''''''''''''
    If IsMissing(Vectors) = True Then
        VectorsToArray = False
        Exit Function
    End If

    ''''''''''''''''''''''''''''''''''''''''''''''
    ' Loop through Vectors to determine the
    ' size of the result array. We do this
    ' loop first to prevent having to do
    ' a Redim Preserve. This requires looping
    ' through Vectors a second time, but this
    ' is still faster than doing Redim Preserves.
    ''''''''''''''''''''''''''''''''''''''''''''''
    For Each Vector In Vectors
        ''''''''''''''''''''''''''''
        ' Ensure Vector is single
        ' dimensional array. This
        ' will take care of the case
        ' if Vector is an unallocated
        ' array (NumberOfArrayDimensions = 0
        ' for an unallocated array).
        ''''''''''''''''''''''''''''
        If NumberOfArrayDimensions(Vector) <> 1 Then
            VectorsToArray = False
            Exit Function
        End If
        ''''''''''''''''''''''''''''''''''''
        ' Ensure that Vector is not an array.
        ''''''''''''''''''''''''''''''''''''
        If IsArray(Vector) = False Then
            VectorsToArray = False
            Exit Function
        End If
        '''''''''''''''''''''''''''''''''
        ' Increment the number of rows.
        ' Each Vector is one row or the
        ' result array. Test the size
        ' of Vector. If it is larger
        ' than the existing value of
        ' NumCols, set NumCols to the
        ' new, larger, value.
        '''''''''''''''''''''''''''''''''
        NumRows = NumRows + 1
        If NumCols < UBound(Vector) - LBound(Vector) + 1 Then
            NumCols = UBound(Vector) - LBound(Vector) + 1
        End If
    Next Vector
    '''''''''''''''''''''''''''''''''''''''''''
    ' Redim Arr to the appropriate size. Arr
    ' is 0-based in both directions, regardless
    ' of the LBound of the original Arr and
    ' regardless of the LBounds of the Vectors.
    ''''''''''''''''''''''''''''''''''''''''''
    ReDim Arr(0 To NumRows - 1, 0 To NumCols - 1)

    ''''''''''''''''''''''''''''''
    ' Loop row-by-row.
    For RowNdx = 0 To NumRows - 1
        ''''''''''''''''''''''''''''''''
        ' Loop through the columns.
        ''''''''''''''''''''''''''''''''
        For ColNdx = 0 To NumCols - 1
            ''''''''''''''''''''''''''''
            ' Set Vector (a Variant) to
            ' the Vectors(RowNdx) array.
            ' We declare Vector as a
            ' variant so it can take an
            ' array of any simple data
            ' type.
            ''''''''''''''''''''''''''''
            Vector = Vectors(RowNdx)
            '''''''''''''''''''''''''''''
            ' The vectors need not ber
            If ColNdx < UBound(Vector) - LBound(Vector) + 1 Then
                VType = VarType(Vector(LBound(Vector) + ColNdx))
                If VType >= vbArray Then
                    ''''''''''''''''''''''''''''''''''''''''''''''''''''''
                    ' Test for VType >= vbArray. The VarType of an array
                    ' is vbArray + VarType(element of array). E.g., the
                    ' VarType of an array of Longs equal vbArray + vbLong.
                    ' Anything greater than or equal to vbArray is an
                    ' array of some time.
                    ''''''''''''''''''''''''''''''''''''''''''''''''''''''
                    VectorsToArray = False
                    Exit Function
                End If
                If VType = vbObject Then
                    VectorsToArray = False
                    Exit Function
                End If
                ''''''''''''''''''''''''''''''''''''''''''''''''''''''''
                ' Vector(LBound(Vector) + ColNdx) is
                ' a simple data type. If Vector(LBound(Vector) + ColNdx)
                ' is not a compatible data type with Arr, then a Type
                ' Mismatch error will occur. We do NOT trap this error.
                ''''''''''''''''''''''''''''''''''''''''''''''''''''''''
                Arr(RowNdx, ColNdx) = Vector(LBound(Vector) + ColNdx)
            End If
        Next ColNdx
    Next RowNdx

    VectorsToArray = True

    End Function

    Public Function IsArraySorted(TestArray As Variant, _
        Optional Descending As Boolean = False) As Variant
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' IsArraySorted
    ' This function determines whether a single-dimensional array is sorted. Because
    ' sorting is an expensive operation, especially so on large array of Variants,
    ' you may want to determine if an array is already in sorted order prior to
    ' doing an actual sort.
    ' This function returns True if an array is in sorted order (either ascending or
    ' descending order, depending on the value of the Descending parameter -- default
    ' is false = Ascending). The decision to do a string comparison (with StrComp) or
    ' a numeric comparison (with < or >) is based on the data type of the first
    ' element of the array.
    ' If TestArray is not an array, is an unallocated dynamic array, or has more than
    ' one dimension, or the VarType of TestArray is not compatible, the function
    ' returns NULL.
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

    Dim StrCompResultFail As Long
    Dim NumericResultFail As Boolean
    Dim Ndx As Long
    Dim NumCompareResult As Boolean
    Dim StrCompResult As Long

    Dim IsString As Boolean
    Dim VType As VbVarType

    ''''''''''''''''''''''''''''''''''
    ' Ensure TestArray is an array.
    ''''''''''''''''''''''''''''''''''
    If IsArray(TestArray) = False Then
        IsArraySorted = Null
        Exit Function
    End If

    ''''''''''''''''''''''''''''''''''''''''''''
    ' Ensure we have a single dimensional array.
    ''''''''''''''''''''''''''''''''''''''''''''
    If NumberOfArrayDimensions(Arr:=TestArray) <> 1 Then
        IsArraySorted = Null
        Exit Function
    End If

    '''''''''''''''''''''''''''''''''''''''''''''
    ' The following code sets the values of
    ' comparison that will indicate that the
    ' array is unsorted. It the result of
    ' StrComp (for strings) or ">=" (for
    ' numerics) equals the value specified
    ' below, we know that the array is
    ' unsorted.
    ''''''''''''''''''''''''''''''''''''''''''''''
    If Descending = True Then
        StrCompResultFail = -1
        NumericResultFail = False
    Else
        StrCompResultFail = 1
        NumericResultFail = True
    End If

    ''''''''''''''''''''''''''''''''''''''''''''''
    ' Determine whether we are going to do a string
    ' comparison or a numeric comparison.
    ''''''''''''''''''''''''''''''''''''''''''''''
    VType = VarType(TestArray(LBound(TestArray)))
    Select Case VType
        Case vbArray, vbDataObject, vbEmpty, vbError, vbNull, vbObject, vbUserDefinedType
        '''''''''''''''''''''''''''''''''
        ' Unsupported types. Reutrn Null.
        '''''''''''''''''''''''''''''''''
            IsArraySorted = Null
            Exit Function
        Case vbString, vbVariant
        '''''''''''''''''''''''''''''''''
        ' Compare as string
        '''''''''''''''''''''''''''''''''
            IsString = True
        Case Else
        '''''''''''''''''''''''''''''''''
        ' Compare as numeric
        '''''''''''''''''''''''''''''''''
            IsString = False
    End Select

    For Ndx = LBound(TestArray) To UBound(TestArray) - 1
        If IsString = True Then
            StrCompResult = StrComp(TestArray(Ndx), TestArray(Ndx + 1))
            If StrCompResult = StrCompResultFail Then
                IsArraySorted = False
                Exit Function
            End If
        Else
            NumCompareResult = (TestArray(Ndx) >= TestArray(Ndx + 1))
            If NumCompareResult = NumericResultFail Then
                IsArraySorted = False
                Exit Function
            End If
        End If
    Next Ndx


    ''''''''''''''''''''''''''''
    ' If we made it out of  the
    ' loop, then the array is
    ' in sorted order. Return
    ' True.
    ''''''''''''''''''''''''''''
    IsArraySorted = True

    End Function

    Public Function CombineTwoDArrays(Arr1 As Variant, _
        Arr2 As Variant) As Variant
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' TwoArraysToOneArray
    ' This takes two 2-dimensional arrays, Arr1 and Arr2, and
    ' returns an array combining the two. The number of Rows
    ' in the result is NumRows(Arr1) + NumRows(Arr2). Arr1 and
    ' Arr2 must have the same number of columns, and the result
    ' array will have that many columns. All the LBounds must
    ' be the same. E.g.,
    ' The following arrays are legal:
    '        Dim Arr1(0 To 4, 0 To 10)
    '        Dim Arr2(0 To 3, 0 To 10)
    '
    ' The following arrays are illegal
    '        Dim Arr1(0 To 4, 1 To 10)
    '        Dim Arr2(0 To 3, 0 To 10)
    '
    ' The returned result array is Arr1 with additional rows
    ' appended from Arr2. For example, the arrays
    '    a    b        and     e    f
    '    c    d                g    h
    ' become
    '    a    b
    '    c    d
    '    e    f
    '    g    h
    '
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

    ''''''''''''''''''''''''''''''''
    ' Upper and lower bounds of Arr1.
    ''''''''''''''''''''''''''''''''
    Dim LBoundRow1 As Long
    Dim UBoundRow1 As Long
    Dim LBoundCol1 As Long
    Dim UBoundCol1 As Long

    '''''''''''''''''''''''''''''''''
    ' Upper and lower bounds of Arr2.
    '''''''''''''''''''''''''''''''''
    Dim LBoundRow2 As Long
    Dim UBoundRow2 As Long
    Dim LBoundCol2 As Long
    Dim UBoundCol2 As Long

    '''''''''''''''''''''''''''''''''''
    ' Upper and lower bounds of Result.
    '''''''''''''''''''''''''''''''''''
    Dim LBoundRowResult As Long
    Dim UBoundRowResult As Long
    Dim LBoundColResult As Long
    Dim UBoundColResult As Long

    '''''''''''''''''
    ' Index Variables
    '''''''''''''''''
    Dim RowNdx1 As Long
    Dim ColNdx1 As Long
    Dim RowNdx2 As Long
    Dim ColNdx2 As Long
    Dim RowNdxResult As Long
    Dim ColNdxResult As Long


    '''''''''''''
    ' Array Sizes
    '''''''''''''
    Dim NumRows1 As Long
    Dim NumCols1 As Long

    Dim NumRows2 As Long
    Dim NumCols2 As Long

    Dim NumRowsResult As Long
    Dim NumColsResult As Long

    Dim Done As Boolean
    Dim Result() As Variant
    Dim ResultTrans() As Variant

    Dim V As Variant


    '''''''''''''''''''''''''''''''
    ' Ensure that Arr1 and Arr2 are
    ' arrays.
    ''''''''''''''''''''''''''''''
    If (IsArray(Arr1) = False) Or (IsArray(Arr2) = False) Then
        CombineTwoDArrays = Null
        Exit Function
    End If

    ''''''''''''''''''''''''''''''''''
    ' Ensure both arrays are allocated
    ' two dimensional arrays.
    ''''''''''''''''''''''''''''''''''
    If (NumberOfArrayDimensions(Arr1) <> 2) Or (NumberOfArrayDimensions(Arr2) <> 2) Then
        CombineTwoDArrays = Null
        Exit Function
    End If

    '''''''''''''''''''''''''''''''''''''''
    ' Ensure that the LBound and UBounds
    ' of the second dimension are the
    ' same for both Arr1 and Arr2.
    '''''''''''''''''''''''''''''''''''''''

    ''''''''''''''''''''''''''
    ' Get the existing bounds.
    ''''''''''''''''''''''''''
    LBoundRow1 = LBound(Arr1, 1)
    UBoundRow1 = UBound(Arr1, 1)

    LBoundCol1 = LBound(Arr1, 2)
    UBoundCol1 = UBound(Arr1, 2)

    LBoundRow2 = LBound(Arr2, 1)
    UBoundRow2 = UBound(Arr2, 1)

    LBoundCol2 = LBound(Arr2, 2)
    UBoundCol2 = UBound(Arr2, 2)

    ''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Get the total number of rows for the result
    ' array.
    ''''''''''''''''''''''''''''''''''''''''''''''''''
    NumRows1 = UBoundRow1 - LBoundRow1 + 1
    NumCols1 = UBoundCol1 - LBoundCol1 + 1
    NumRows2 = UBoundRow2 - LBoundRow2 + 1
    NumCols2 = UBoundCol2 - LBoundCol2 + 1

    '''''''''''''''''''''''''''''''''''''''''
    ' Ensure the number of columns are equal.
```

```
''''''''''''''''''''''''''''''''''''''''''
If NumCols1 <> NumCols2 Then
    CombineTwoDArrays = Null
    Exit Function
End If

NumRowsResult = NumRows1 + NumRows2

''''''''''''''''''''''''''''''''''''''''
' Ensure that ALL the LBounds are equal.
'''''''''''''''''''''''''''''''''''''''''
If (LBoundRow1 <> LBoundRow2) Or _
    (LBoundRow1 <> LBoundCol1) Or _
    (LBoundRow1 <> LBoundCol2) Then
    CombineTwoDArrays = Null
    Exit Function
End If
'''''''''''''''''''''''''''''''''
' Get the LBound of the columns
' of the result array.
'''''''''''''''''''''''''''''''''
LBoundColResult = LBoundRow1
'''''''''''''''''''''''''''''''''
' Get the UBound of the columns
' of the result array.
'''''''''''''''''''''''''''''''''
UBoundColResult = UBoundCol1

UBoundRowResult = LBound(Arr1, 1) + NumRows1 + NumRows2 - 1
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Redim the Result array to have number of rows equal to
' number-of-rows(Arr1) + number-of-rows(Arr2)
' and number-of-columns equal to number-of-columns(Arr1)
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
ReDim Result(LBoundRow1 To UBoundRowResult, LBoundColResult To UBoundColResult)

RowNdxResult = LBound(Result, 1) - 1

Done = False
Do Until Done
    '''''''''''''''''''''''''''''''''''''''''''''
    ' Copy elements of Arr1 to Result
    ''''''''''''''''''''''''''''''''''''''''''''
    For RowNdx1 = LBound(Arr1, 1) To UBound(Arr1, 1)
        RowNdxResult = RowNdxResult + 1
        For ColNdx1 = LBound(Arr1, 2) To UBound(Arr1, 2)
            V = Arr1(RowNdx1, ColNdx1)
            Result(RowNdxResult, ColNdx1) = V
        Next ColNdx1
    Next RowNdx1

    '''''''''''''''''''''''''''''''''''''''''''''
    ' Copy elements of Arr2 to Result
    '''''''''''''''''''''''''''''''''''''''''''''
    For RowNdx2 = LBound(Arr2, 1) To UBound(Arr2, 1)
        RowNdxResult = RowNdxResult + 1
        For ColNdx2 = LBound(Arr2, 2) To UBound(Arr2, 2)
            V = Arr2(RowNdx2, ColNdx2)
            Result(RowNdxResult, ColNdx2) = V
        Next ColNdx2
    Next RowNdx2

    If RowNdxResult >= UBound(Result, 1) + (LBoundColResult = 1) Then
        Done = True
    End If
'''''''''''''
' End Of Loop
'''''''''''''
Loop
'''''''''''''''''''''''''
' Return the Result
'''''''''''''''''''''''''
CombineTwoDArrays = Result

End Function

Function ExpandArray(Arr As Variant, WhichDim As Long, AdditionalElements As Long, _
        FillValue As Variant) As Variant
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' ExpandArray
' This expands a two-dimensional array in either dimension. It returns the result
' array if successful, or NULL if an error occurred. The original array is never
' changed.
' Paramters:
' --------------------
' Arr                   is the array to be expanded.
'
' WhichDim              is either 1 for additional rows or 2 for
'                       additional columns.
'
' AdditionalElements    is the number of additional rows or columns
'                       to create.
'
' FillValue             is the value to which the new array elements should be
'                       initialized.
'
' You can nest calls to Expand array to expand both the number of rows and
' columns. E.g.,
'
' C = ExpandArray(ExpandArray(Arr:=A, WhichDim:=1, AdditionalElements:=3, FillValue:="R"), _
'    WhichDim:=2, AdditionalElements:=4, FillValue:="C")
' This first adds three rows at the bottom of the array, and then adds four
' columns on the right of the array.
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim Result As Variant
Dim RowNdx As Long
Dim ColNdx As Long
Dim ResultRowNdx As Long
Dim ResultColNdx As Long
Dim NumRows As Long
Dim NumCols As Long
Dim NewUBound As Long

Const ROWS_ As Long = 1
Const COLS_ As Long = 2


''''''''''''''''''''''''''''
' Ensure Arr is an array.
''''''''''''''''''''''''''''
If IsArray(Arr) = False Then
    ExpandArray = Null
    Exit Function
End If
'''''''''''''''''''''''''''''''''
' Ensure Arr has two dimenesions.
'''''''''''''''''''''''''''''''''
If NumberOfArrayDimensions(Arr:=Arr) <> 2 Then
    ExpandArray = Null
    Exit Function
End If
'''''''''''''''''''''''''''''''''
' Ensure the dimension is 1 or 2.
'''''''''''''''''''''''''''''''''
Select Case WhichDim
    Case 1, 2
    Case Else
        ExpandArray = Null
        Exit Function
End Select

''''''''''''''''''''''''''''''''''''
' Ensure AdditionalElements is > 0.
' If AdditionalElements  < 0, return NULL.
' If AdditionalElements  = 0, return Arr.
''''''''''''''''''''''''''''''''''''
If AdditionalElements < 0 Then
    ExpandArray = Null
    Exit Function
End If
If AdditionalElements = 0 Then
    ExpandArray = Arr
    Exit Function
End If

NumRows = UBound(Arr, 1) - LBound(Arr, 1) + 1
NumCols = UBound(Arr, 2) - LBound(Arr, 2) + 1

If WhichDim = ROWS_ Then
    '''''''''''''''
    ' Redim Result.
    '''''''''''''''
    ReDim Result(LBound(Arr, 1) To UBound(Arr, 1) + AdditionalElements, LBound(Arr, 2) To UBound(Arr, 2))
    ''''''''''''''''''''''''''''''
    ' Transfer Arr array to Result
    ''''''''''''''''''''''''''''''
    For RowNdx = LBound(Arr, 1) To UBound(Arr, 1)
        For ColNdx = LBound(Arr, 2) To UBound(Arr, 2)
            Result(RowNdx, ColNdx) = Arr(RowNdx, ColNdx)
        Next ColNdx
    Next RowNdx
    '''''''''''''''''''''''''''''''
    ' Fill the rest of the result
    ' array with FillValue.
    '''''''''''''''''''''''''''''''
    For RowNdx = UBound(Arr, 1) + 1 To UBound(Result, 1)
        For ColNdx = LBound(Arr, 2) To UBound(Arr, 2)
            Result(RowNdx, ColNdx) = FillValue
        Next ColNdx
    Next RowNdx
Else
    '''''''''''''''
    ' Redim Result.
    '''''''''''''''
    ReDim Result(LBound(Arr, 1) To UBound(Arr, 1), UBound(Arr, 2) + AdditionalElements)
    ''''''''''''''''''''''''''''''
    ' Transfer Arr array to Result
    ''''''''''''''''''''''''''''''
    For RowNdx = LBound(Arr, 1) To UBound(Arr, 1)
        For ColNdx = LBound(Arr, 2) To UBound(Arr, 2)
            Result(RowNdx, ColNdx) = Arr(RowNdx, ColNdx)
        Next ColNdx
    Next RowNdx
    '''''''''''''''''''''''''''''''
    ' Fill the rest of the result
    ' array with FillValue.
    '''''''''''''''''''''''''''''''
    For RowNdx = LBound(Arr, 1) To UBound(Arr, 1)
        For ColNdx = UBound(Arr, 2) + 1 To UBound(Result, 2)
```

```
            Result(RowNdx, ColNdx) = FillValue
        Next ColNdx
    Next RowNdx

End If
''''''''''''''''''''
' Return the result.
''''''''''''''''''''
ExpandArray = Result

End Function

Function SwapArrayRows(Arr As Variant, Row1 As Long, Row2 As Long) As Variant
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' SwapArrayRows
' This function returns an array based on Arr with Row1 and Row2 swapped.
' It returns the result array or NULL if an error occurred.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim V As Variant
Dim Result As Variant
Dim RowNdx As Long
Dim ColNdx As Long

'''''''''''''''''''''''''
' Ensure Arr is an array.
'''''''''''''''''''''''''
If IsArray(Arr) = False Then
    SwapArrayRows = Null
    Exit Function
End If

''''''''''''''''''''''''''''''''
' Set Result to Arr
''''''''''''''''''''''''''''''''
Result = Arr

''''''''''''''''''''''''''''''''
' Ensure Arr is two-dimensional.
''''''''''''''''''''''''''''''''
If NumberOfArrayDimensions(Arr:=Arr) <> 2 Then
    SwapArrayRows = Null
    Exit Function
End If

''''''''''''''''''''''''''''''''
' Ensure Row1 and Row2 are less
' than or equal to the number of
' rows.
''''''''''''''''''''''''''''''''
If (Row1 > UBound(Arr, 1)) Or (Row2 > UBound(Arr, 1)) Then
    SwapArrayRows = Null
    Exit Function
End If

'''''''''''''''''''''''''''''''''
' If Row1 = Row2, just return the
' array and exit. Nothing to do.
'''''''''''''''''''''''''''''''''
If Row1 = Row2 Then
    SwapArrayRows = Arr
    Exit Function
End If

'''''''''''''''''''''''''''''''''''''''''
' Redim V to the number of columns.
'''''''''''''''''''''''''''''''''''''''''
ReDim V(LBound(Arr, 2) To UBound(Arr, 2))
'''''''''''''''''''''''''''''''''''''''''
' Put Row1 in V
'''''''''''''''''''''''''''''''''''''''''
For ColNdx = LBound(Arr, 2) To UBound(Arr, 2)
    V(ColNdx) = Arr(Row1, ColNdx)
    Result(Row1, ColNdx) = Arr(Row2, ColNdx)
    Result(Row2, ColNdx) = V(ColNdx)
Next ColNdx

SwapArrayRows = Result

End Function


Function SwapArrayColumns(Arr As Variant, Col1 As Long, Col2 As Long) As Variant
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' SwapArrayColumns
' This function returns an array based on Arr with Col1 and Col2 swapped.
' It returns the result array or NULL if an error occurred.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim V As Variant
Dim Result As Variant
Dim RowNdx As Long
Dim ColNdx As Long

'''''''''''''''''''''''''
' Ensure Arr is an array.
'''''''''''''''''''''''''
If IsArray(Arr) = False Then
    SwapArrayColumns = Null
    Exit Function
End If
```

```
''''''''''''''''''''''''''''''''
' Set Result to Arr
''''''''''''''''''''''''''''''''
Result = Arr

''''''''''''''''''''''''''''''''
' Ensure Arr is two-dimensional.
''''''''''''''''''''''''''''''''
If NumberOfArrayDimensions(Arr:=Arr) <> 2 Then
    SwapArrayColumns = Null
    Exit Function
End If

''''''''''''''''''''''''''''''''
' Ensure Row1 and Row2 are less
' than or equal to the number of
' rows.
''''''''''''''''''''''''''''''''
If (Col1 > UBound(Arr, 2)) Or (Col2 > UBound(Arr, 2)) Then
    SwapArrayColumns = Null
    Exit Function
End If

'''''''''''''''''''''''''''''''''
' If Col1 = Col2, just return the
' array and exit. Nothing to do.
'''''''''''''''''''''''''''''''''
If Col1 = Col2 Then
    SwapArrayColumns = Arr
    Exit Function
End If

'''''''''''''''''''''''''''''''''''''''''
' Redim V to the number of columns.
'''''''''''''''''''''''''''''''''''''''''
ReDim V(LBound(Arr, 1) To UBound(Arr, 1))
'''''''''''''''''''''''''''''''''''''''''
' Put Col2 in V
'''''''''''''''''''''''''''''''''''''''''
For RowNdx = LBound(Arr, 1) To UBound(Arr, 1)
    V(RowNdx) = Arr(RowNdx, Col1)
    Result(RowNdx, Col1) = Arr(RowNdx, Col2)
    Result(RowNdx, Col2) = V(RowNdx)
Next RowNdx

SwapArrayColumns = Result

End Function

Function GetColumn(Arr As Variant, ResultArr As Variant, ColumnNumber As Long) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' GetColumn
' This populates ResultArr with a one-dimensional array that is the
' specified column of Arr. The existing contents of ResultArr are
' destroyed. ResultArr must be a dynamic array.
' Returns True or False indicating success.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim RowNdx As Long
''''''''''''''''''''''''''''''
' Ensure Arr is an array.
''''''''''''''''''''''''''''''
If IsArray(Arr) = False Then
    GetColumn = False
    Exit Function
End If

''''''''''''''''''''''''''''''''''
' Ensure Arr is a two-dimensional
' array.
''''''''''''''''''''''''''''''''''
If NumberOfArrayDimensions(Arr) <> 2 Then
    GetColumn = False
    Exit Function
End If

''''''''''''''''''''''''''''''''''
' Ensure ResultArr is a dynamic
' array.
''''''''''''''''''''''''''''''''''
If IsArrayDynamic(ResultArr) = False Then
    GetColumn = False
    Exit Function
End If

''''''''''''''''''''''''''''''''''''
' Ensure ColumnNumber is less than
' or equal to the number of columns.
''''''''''''''''''''''''''''''''''''
If UBound(Arr, 2) < ColumnNumber Then
    GetColumn = False
    Exit Function
End If
If LBound(Arr, 2) > ColumnNumber Then
    GetColumn = False
    Exit Function
End If

Erase ResultArr
ReDim ResultArr(LBound(Arr, 1) To UBound(Arr, 1))
For RowNdx = LBound(ResultArr) To UBound(ResultArr)
```

```
    ResultArr(RowNdx) = Arr(RowNdx, ColumnNumber)
Next RowNdx

GetColumn = True


End Function


Function GetRow(Arr As Variant, ResultArr As Variant, RowNumber As Long) As Boolean
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' GetRow
' This populates ResultArr with a one-dimensional array that is the
' specified row of Arr. The existing contents of ResultArr are
' destroyed. ResultArr must be a dynamic array.
' Returns True or False indicating success.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim ColNdx As Long
''''''''''''''''''''''''''''''
' Ensure Arr is an array.
''''''''''''''''''''''''''''''
If IsArray(Arr) = False Then
    GetRow = False
    Exit Function
End If

''''''''''''''''''''''''''''''''''
' Ensure Arr is a two-dimensional
' array.
''''''''''''''''''''''''''''''''''
If NumberOfArrayDimensions(Arr) <> 2 Then
    GetRow = False
    Exit Function
End If

''''''''''''''''''''''''''''''''''
' Ensure ResultArr is a dynamic
' array.
''''''''''''''''''''''''''''''''''
If IsArrayDynamic(ResultArr) = False Then
    GetRow = False
    Exit Function
End If

''''''''''''''''''''''''''''''''''''
' Ensure ColumnNumber is less than
' or equal to the number of columns.
''''''''''''''''''''''''''''''''''''
If UBound(Arr, 1) < RowNumber Then
    GetRow = False
    Exit Function
End If
If LBound(Arr, 1) > RowNumber Then
    GetRow = False
    Exit Function
End If

Erase ResultArr
ReDim ResultArr(LBound(Arr, 2) To UBound(Arr, 2))
For ColNdx = LBound(ResultArr) To UBound(ResultArr)
    ResultArr(ColNdx) = Arr(RowNumber, ColNdx)
Next ColNdx

GetRow = True


End Function
```