Parsing Strings

To parse a string means to extract subsections of the string based on their position or, more commonly, a delimiter character. Common needs for parsing strings include using command line arguments and reading CSV (comma separated value) files. This tip presents a Visual Basic function that should meet all your parsing needs.

On the surface, parsing a string based on a delimiter character seems simple enough. For example, the string "a,b,c" is easily parsed into the three substrings "a" "b" and "c" based on using the comma as the delimiter character. But things are not always so simple. What about the string "a,,b,c"? This should be parsed as four substrings because the pair of commas indicates a missing, or blank, second substring. The same is true of the string ",a,b,c" where the initial comma indicates a missing or blank first substring. Thus, any parsing function must be able to take such missing or blank values into account.

A second requirements is the use of multiple characters as a delimiter. In some applications this may be desirable so as to not preclude the use of any characters in the data. For example, using the \ character as a delimiter works fine but means that the data itself cannot contain this character. Using \\ will circumvent this limitation.

The function ParseString() meets all of these requirements. It is passed two arguments, the string to be parsed and the delimiter string. It returns an array containing the parsed substring. Under certain error conditions (explained in the comments in the code) it returns Null. The calling program can use the UBound function to determine the size of the returned array and then access the individual substrings as needed.

```
Public Function ParseString(StringToParse As String, Delim As String) As
Variant

' Returns an array containing the results of parsing
' a string based on the specified delimiter.
' If StringToParse or Delim is empty returns Null
' If Len(Delim) >= Len(StringToParse returns Null.
' If StringToParse does not contain the delimiter, returns
' the entire string.
```

```
Dim a() As String
Dim s As String
Dim DelimPos As Integer
Dim count As Integer

If Len(StringToParse) = 0 Or Len(Delim) = 0 Then
   ParseString = Null
   Exit Function
End If

If Len(Delim) >= Len(StringToParse) Then
   ParseString = Null
   Exit Function
End If

DelimPos = InStr(1, StringToParse, Delim)
If DelimPos = 0 Then
   ReDim a(0)
   a(0) = StringToParse
   ParseString = a
   Exit Function
End If

s = StringToParse
count = 0

Do
   ReDim Preserve a(count)
   a(count) = Left(s, DelimPos - 1)
   s = Right(s, Len(s) - (DelimPos + Len(Delim) - 1))
   count = count + 1
   DelimPos = InStr(1, s, Delim)
   If DelimPos = 0 Then
      ReDim Preserve a(count)
      a(count) = s
      s = ""
   End If
Loop Until Len(s) = 0

ParseString = a
```

End Function