

#### What You Will Need

- ArcGIS 8.2 or higher (ArcView, ArcEditor, or ArcInfo license)
- Sample data downloaded from *ArcUser Online*

# Using VBScript to Build Complex Labels in ArcGIS

By Chip Hankley, RMT, Inc.

The ability to dynamically retrieve attribute information associated with geographic features sets GIS apart from other mapping technologies. Labels can be automatically generated, reducing the risk of human error associated with typing and transcription. While simple labeling in GIS is fairly routine, creating complex labels has traditionally been much more difficult.

In ArcMap, the Expression Builder allows scripting (either JavaScript or VBScript) to be used for preprocessing and building labels on the fly. Because both JavaScript and VBScript are extremely powerful languages, this feature goes a long way toward simplifying the process of creating complex labels. Leveraging the power of these languages to generate labels opens many new opportunities for streamlining the labeling process. Although either language could be used, the examples in this article will be limited to VBScript.

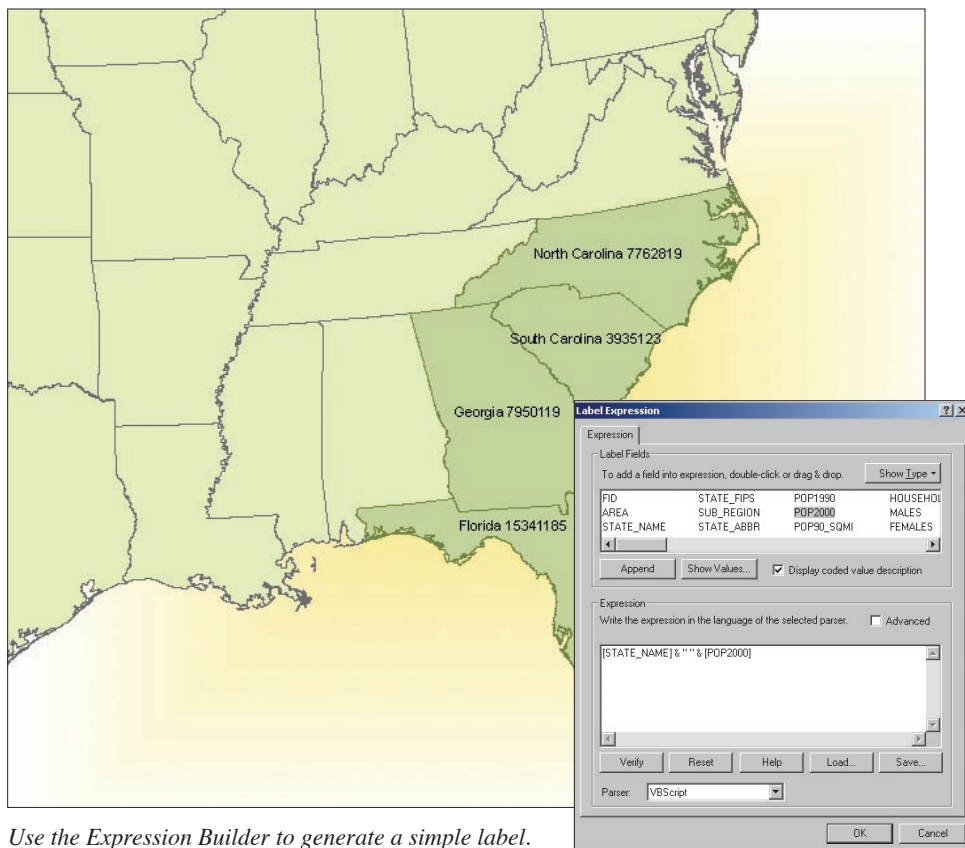
## Data for the Exercise

This article uses the DTL\_ST shapefile that comes in the sample data for ArcGIS on the *ESRI Data & Maps Media Kit* CD-ROMs. This shapefile is on the disc with United States data and represents all 50 U.S. states and the District of Columbia. Shapefile attributes contain general state-by-state census information from the 2000 census. To more easily and clearly illustrate the examples described in this article, a shapefile consisting of just four states called MidAtl.shp was included with the sample dataset. This shapefile will be used for these labeling exercises.

## Simple Expression Builder Example

This basic example shows how scripting works in the Expression Builder.

1. Start an ArcMap session and add the DTL\_ST and MidAtl shapefiles.
2. Right-click the MidAtl layer and choose Properties from the context menu to access the Layer Properties dialog box. Click the Labels tab, then click the Expression button.
3. Delete anything in the Expression section of the dialog box. Under the Layer Fields section, double-click STATE\_NAME, highlight POP2000, and click the Append button. The contents of the dialog box should read: [STATE\_NAME] & "" & [POP2000]
4. Click the Verify button. A box should pop up with an example of the label contents displayed in the following format:



Use the Expression Builder to generate a simple label.

Florida 15341185

5. Click OK twice to dismiss the Properties dialog box. Right-click the MidAtl layer and click Label Features to display the labels using the expression just created.

## Using VBScript and Formatting Tags

VBScript and ArcMap text formatting tags can be used to format the label. The state's name will be capitalized and displayed using the Arial font at 12 points. The population figure will be printed on a new line using the Lucida Console font at 10 points with a thousands separator and will be preceded by the label YEAR 2000 POPULATION. All text in the label will be capitalized. To do this, access the Properties dialog box for STL\_ST as previous-

ly described and click the Expression button. Change the expression so it looks like the code in Figure 1.

When typing this code, note that in to make the text more readable in the Expression Builder dialog box, the Visual Basic (VB) line continuation character, an underscore (\_), was used. Make sure the expression contains no carriage returns. ArcMap text formatting tags follow the basic rules of XML so each expression must be well formed. Pay attention to the following three things when using formatting tags.

- When typing starting and closing tags, you must use the same case for both. For instance "<ACP>" must be closed by "</ACP>", not "</acp>".

```
"<ACP><FNT name='Arial' size='12'>" & [STATE_NAME] & _  
"</FNT>" & vbnewline & _  
"<FNT name='Lucida Console' size='10'>Year 2000 Population: " & _  
formatnumber([POP2000],0,-1,-1,-1) & "</FNT></ACP>"
```

Figure 1: Sample expression code for simple example



The expression used for these labels was built using mostly text formatting tags. The only VBScript component is the “formatnumber” function, which suppresses the decimal places and adds a thousands separator.

- Tags must be nested properly, and inner tags must be closed before outer tags are closed.
- The ampersand (&), greater than (>), and less than (<) characters are illegal in text formatting tags when used as part of a field value in an expression or as part of the label to be rendered. (Note that this does not refer to the “&” used to join the various parts of the expression.) For instance, the state field for a polygon could have a value of “Delaware & Maryland,” or a results field on a chemistry dataset could have a value of “< 0.01 mg/kg.” These characters can be used by replacing them with the following HTML escape characters—&amp; for the ampersand, &gt; for the greater than sign, and &lt; for the less than sign.

Errors in text formatting tag syntax will not cause ArcMap to generate an error; however, when labels are rendered in ArcMap, the formatting tags will not be interpreted and will be rendered as part of the label. Clicking the Verify button will display the raw text that is generated by the ArcMap labeling engine, which means the formatting tags will be visible. To see the labels in final form, view the labels in ArcMap.

The expression in Figure 1 is built mostly using the text formatting tags. The only VBScript component is the FormatNumber function. The syntax for the FormatNumber function is shown in Figure 2.

In this case, the number is formatted with-

```
FormatNumber(Expression [,NumDigitsAfterDecimal [,IncludeLeadingDigit [,UseParensForNegativeNumbers [,GroupDigits]]]])
```

Figure 2: Syntax for the FormatNumber function

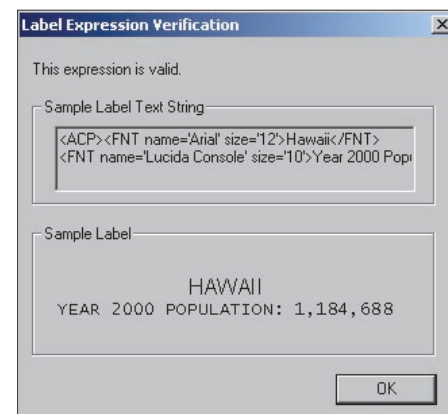
```
Function FindLabel([field1],[field2],..., [fieldx])
...some code
FindLabel = ...something
End Function
```

Figure 3: Performing more advanced labeling

out decimal places and uses a thousands separator. To do this, a “0” is passed for the second argument and a “-1” is passed for the fifth argument. Arguments were passed for the third and fourth elements, but that is irrelevant for this dataset because all the values are greater than 1. Clicking the Help button below the Expression section in the dialog box will bring up the ArcGIS Help topic “About building label expressions,” which contains a link to the Microsoft VBScript Language Reference. This reference is invaluable for learning what is available in VBScript and for looking up syntax.

#### Advanced Labeling With Logic

Sometimes more complex labeling requires logic that will allow the labeling routine to do



The Verify button in the Label Expression dialog box is used by ArcMap to validate the label expression. ArcMap will generate an error message in response to scripting errors. If no error is found, the message box indicates the expression is valid and provides an example of the label text. Any text formatting tags are rendered as raw text in the sample label dialog box.

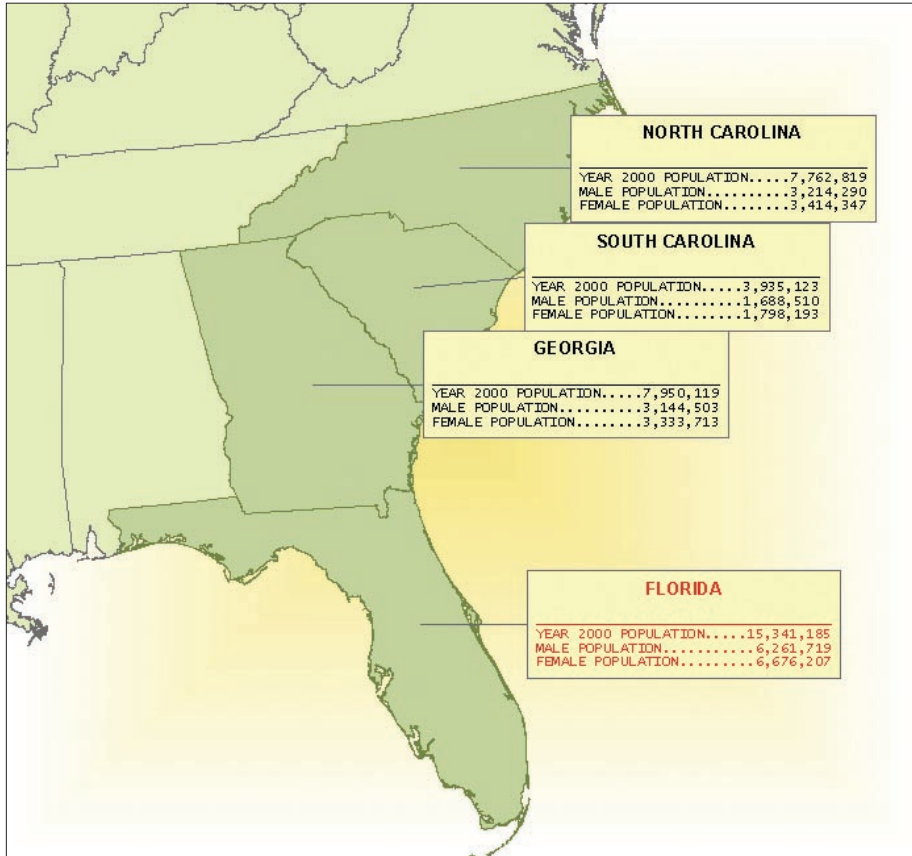
something different each time based on the condition of the data. In such cases, use the Advanced Expression Builder. To access the Advanced Expression Builder, check the Advanced check box in the Expression section of the Label Expression dialog box. The contents of the Expression section will change to a function written in the language of the parser that is selected at the bottom of the dialog box. The format of the function will always look like the code shown in Figure 3.

Note that in the first line, where the function is declared, the field name(s) that will be used in the label as arguments need to be passed. Field names need to be placed in square brackets, separated by commas. The value of the variable FindLabel is what is displayed on the

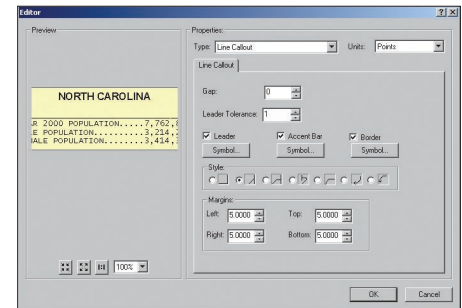
*Continued on page 52*

# Using VBScript to Build Complex Labels in ArcGIS

Continued from page 51



*Dynamically generating complex labels has traditionally been a complex task. Leveraging the power of VBScript or JavaScript within the Label Expression dialog box opens virtually unlimited opportunities for new label types.*



*As a final step, the Layer Properties dialog box will be used to format the labels with a solid background and a leader line.*

map so the value of the variable FindLabel will need to be set inside the function. By using a function, the full power of the scripting language can be leveraged by building real-time logic (in the form of a small program) into the label rendering process.

## An Advanced Example—Stacked Labels

The script in Figure 4 shows the VBScript routine that was developed to build a multilined (i.e., stacked) label that will have the following characteristics:

- The state's name will be capitalized and displayed in bold Arial font at 10 points.
- Subsequent lines will show values for the POP2000, MALES, and FEMALES fields. Each line will use the Lucida Console font at 8 points. The left-hand side of the line will be an intelligent field label (i.e., the label field will display as Year 2000 Population rather than POP2000). The right-hand side of the line will have the value of the field formatted using a thousands separator. The right- and left-hand parts of each line will be separated by repeating periods, and each line will have the same character count. Because Lucida Console is a fixed-width font, this will give the labels a uniform, almost table-like appearance. These lines will be separated from the first line by a solid horizontal line.

The color of the label will be based on

the state's population. States that have a 2000 census population greater than 10 million will have red text. All others will have black text.

1. Access the Advanced Expression Builder, delete any existing text, and type or cut and paste the code in Figure 4 into the Label Expression dialog box. Alternatively, click the Load button, navigate to the sample data, and select AdvancedLabel.lxp, a saved copy of this script. Click the Verify button.
2. Click OK to return to the Layer Properties dialog box. As a final step, the Layer Properties dialog box will be used to format the labels with a solid background and a leader line. From the Label tab, choose the Symbol button.
3. This brings up the Symbol Selector dialog box. Click the Properties button. Click the General tab, make the horizontal alignment Center, and set the X Offset to 75,000.
4. Next, choose the Advanced Text tab. Check the box next to Text Background and click the Properties button. In the Editor dialog box, choose Line Callout as the Type. Set Gap to 0 and Leader Tolerance to 1, and click the second choice from the left under Style. Close all dialog boxes, zoom the map scale to 1:9,200,000, and refresh the map display.

## Conclusion

This sample script only scratches the surface of

what is possible using a scripting language to develop labels. If needed, subroutines can be included in the expression code. For example, a subroutine could be written that would check for the illegal characters <, >, and & and replace them with the appropriate escape characters.

Developing a script that creates advanced labels can be a complex task, but once created, such a script provides a scalable and reusable platform for future versions of the map. When labeling needs change, the script can be easily modified to address the new requirements and new labels can be generated automatically. This process is much less painful than the traditional label-by-label editing process.

For more information, contact  
Chip Hankley, GIS Specialist  
RMT, Inc.  
744 Heartland Trail  
Madison, Wisconsin 53717-1934  
E-mail: [chip.hankley@rmtinc.com](mailto:chip.hankley@rmtinc.com)

## About the Author

Chip Hankley is a GIS specialist at RMT, Inc., a global environmental and engineering firm. He specializes in the use of GIS for natural resource management and environmental cleanup. He has a master's degree in environmental engineering from Old Dominion University in Norfolk, Virginia.

```

Function FindLabel([STATE_NAME], [POP2000], [MALES], [FEMALES])
    iMxLblSz = 0 'The MAX width of the descriptive label
    iMxValSz = 0 'The MAX width of the values
    iSpace = 5 'The minimum number of spaces (periods)
                'between a label and a value

    'Build a nested array where each sub-array contains the descriptive name of
    'the field and the field value
    m = Array(Array("Year 2000 Population", [POP2000]), _
        Array("Male Population", [MALES]), _
        Array("Female Population", [FEMALES]))

    'Calculate the values for iMxLblSz and iMxValSz by looping through
    'the array and calculating the length of each element (tracking
    'the descriptive labels and values separately). The calculated size
    'is recorded, then checked against previous values, ultimately
    'identifying the longest (number of characters) value. Note that
    'when calculating lengths, we are calculating the length of the
    'formatted number (i.e. with commas) for the field value.
    For i = 0 To UBound(m)
        j = m(i)
        If (Len(j(0)) > iMxLblSz) Then
            iMxLblSz = Len(j(0))
        End If
        If (Len(FormatNumber(j(1), 0, 0, 0, -1)) > iMxValSz) Then
            iMxValSz = Len(FormatNumber(j(1), 0, 0, 0, -1))
        End If
    Next

    'START BUILDING THE LABEL
    'Make tags to format the label so that it's red if the population is
    'greater than 10 Million, otherwise make the label black.
    j = m(0) 'Re-set j to be the first sub-array in the array m
    If (j(1) > 10000000) Then
        strGT10M1 = "<CLR red='255' green='0' blue='0'>"
    Else
        strGT10M1 = "<CLR red='0' green='0' blue='0'>"
    End If

    'Make a closing tag
    strGT10M2 = "</CLR>"

    'Make the first line with the state name and begin generating the text
    'formatting tags.
    FindLabel = strGT10M1 + "<ACP><BOL><FNT name='Arial' size='10'>" & _
        [STATE_NAME] & "</FNT></BOL>" & vbNewLine & _
        "<FNT name='Lucida Console' size='8'>" & vbNewLine

    'Make a solid line by repeating the underscore character the
    'correct number of times.
    FindLabel = FindLabel & String(iMxLblSz + iMxValSz + iSpace, "_") & _
        vbNewLine

    'Make the subsequent lines by looping through the array. For each
    'line we need to calculate the number of periods that we need to
    'add between the descriptive label and the field value so that
    'each line has the same number of characters. Note that when we
    'do calculations on the field value, we are calculating against
    'the formatted (i.e. with commas) value.
    For i = 0 To UBound(m)
        j = m(i)

        'Calculate the number of periods that need to go between
        'the descriptive label and the value by 1) subtracting the
        'size of each from the maximum size calculated above, 2)
        'adding the minimum number of periods (iSpace).
        k = (iMxLblSz - Len(j(0))) + iSpace + _
            (iMxValSz - Len(FormatNumber(j(1), 0, 0, 0, -1)))

        'Build the label line by adding the field label, the right
        'number of periods, then the number
        FindLabel = FindLabel + j(0) + String(k, ".") + _
            FormatNumber(j(1), 0, 0, 0, -1) + vbNewLine
    Next

    'Close the text formatting tags.
    FindLabel = FindLabel + "</FNT></ACP>" + strGT10M2

End Function

```

Figure 4: Sample expression code for advanced example

Developing a script that creates advanced labels can be a complex task, but once created, such a script provides a scalable and reusable platform for future versions of the map.