July, 2002

Advisor Answers

Checking Spelling in VFP

VFP 7.0

Q: Is it possible to use the spelling checker from Office to check the contents of a VFP editbox?

–Anne-Mie Vanhulle (via DevX.COM)

A: You can do this using Automation to the spelling engine. However, while several Office products use the spelling engine, as far as I can tell, it's not available as a separate object; it can only be used from within one of the other products. (For an explanation of Microsoft's position on this issue, see Microsoft Knowledge Base article Q262605.) Both Word and Excel expose spelling methods, but Word's is better behaved for Automation purposes, so we'll tackle the problem from that side.

Word offers two approaches to checking spelling. First, you can simply pass a string to the CheckSpelling method of the application object and get back a logical value indicating whether the string passes. So, if all you want is to know whether or not there are any spelling problems, you can use code like this:

```
cString = "The last word of this sentence is mispelled"
* or, to check an editbox value, something like:
* cString = ThisForm.edtMyEditBox.Value
oWord = CreateObject("Word.Application")
lCorrect = oWord.CheckSpelling(cString)
```

However, in most cases, you probably want more than that. If there is a spelling error, you'd like to know what it is and give the user a chance to do something about it. Fortunately, Word offers another option. The GetSpellingSuggestions method fills a SpellingSuggestions collection with recommended corrections for the word you pass to it. If the collection is empty (its Count property is 0), the word passed the spelling check. However, you must have a document open in Word, even if it's an empty one. So you can check a word and get a list of suggestions like this:

```
cWord = "mispelled"
oWord = CreateObject("Word.Application")
oWord.Documents.Add()
```

```
oSuggestions = oWord.GetSpellingSuggestions( cWord )
```

Then, to see the suggestions, you can use a FOR EACH loop:

```
FOR EACH oSuggestion in oSuggestions
   ? oSuggestion.Name
ENDFOR
```

Of course, in an application, you wouldn't just display the suggestions in the active window. You might handle them by displaying a listbox or by populating a context menu or in any of a number of other ways. I'll leave that part of the problem to you.

However, to make it easy to do whatever you need, I created a class with a method that calls the GetSpellingSuggestions method and puts all the suggestions into an array. Since there might be other facilities from Word you'd like to use in this way, the class library is called WordUtils. This class is cusSpellCheck.

The class has three custom properties:

oWord contains an object reference to Word.

aSuggestions is an array property with one suggested correction per row. Each row contains the original word, its position in the original string, and one suggested correction.

nSuggestions contains the total number of suggestions, that is, the number of rows in aSuggestions.

There are two custom methods, CheckWord and CheckSpelling. CheckWord checks whether oWord currently contains a reference to Word. If not, it starts Word and stores the reference in the oWord property. Here's the code for CheckWord:

```
* Check for a Word object.
* If not found, instantiate one.

LOCAL lReturn

lReturn = .T.
IF VARTYPE(This.oWord)#"O" OR TYPE("This.oWord.Name")#"C"
   This.oWord = CREATEOBJECT( "Word.Application" )
ENDIF

* Double-check
IF VARTYPE(This.oWord)#"O" OR TYPE("This.oWord.Name")#"C"
   lReturn = .F.
ENDIF
```

```
RETURN lReturn
```

CheckSpelling is the heart of the class. You pass it a string and it checks the spelling for each word in the string. All spelling suggestions are stored in the aSuggestions array property. Here's the code:

```
* CheckSpelling
LPARAMETERS cString

ASSERT VARTYPE(cString) = "C" ;
   MESSAGE "CheckSpelling: First parameter " + ;
           "(cString) must be character"

IF VARTYPE(cString) <> "C"
   ERROR 11
   RETURN .F.
ENDIF

LOCAL lReturn, nWords, nWord
LOCAL oSuggestions as Word.SpellingSuggestions
LOCAL oSuggestion as Word.SpellingSuggestion

DIMENSION This.aSuggestions[1]
This.aSuggestions[1] = ""
This.nSuggestionCount = 0

IF EMPTY(cString)
   lReturn = .t.
ELSE
   IF This.CheckWord()
      WITH This.oWord
         .Documents.Add()

         lReturn = .T.
         nWords = GETWORDCOUNT( cString)
         nSuggCount = 0
         FOR nWord = 1 TO nWords
            cWord = GETWORDNUM( cString, nWord )
            oSuggestions = .GetSpellingSuggestions(cWord)
            IF oSuggestions.Count <> 0
               lReturn = .F.
               * Parse the list and put into the array
               FOR EACH oSuggestion IN oSuggestions
                  This.nSuggestionCount = ;
                     This.nSuggestionCount + 1
                  DIMENSION This.aSuggestions[ ;
                     This.nSuggestionCount, 3]
                  This.aSuggestions[ ;
                     This.nSuggestionCount, 1 ] = nWord
                  This.aSuggestions[ ;
                     This.nSuggestionCount, 2 ] = cWord
                  This.aSuggestions[ ;
                     This.nSuggestionCount, 3 ] = ;
                     oSuggestion.Name
               ENDFOR
            ENDIF
         ENDFOR
      ENDWITH
   ELSE
      lReturn = .F.
   ENDIF
ENDIF

RETURN lReturn
```

Finally, there's code in the Destroy method to close the Word instance:

```
* Clean up

IF VARTYPE(This.oWord) = "O"
   This.oWord.Quit(0)
ENDIF

RETURN
```

To use the class to check spelling, use code like this:

```
cString = "The last word of this sentence is mispelled"
oSpeller = NewObject("cusSpellCheck","WordUtils")
IF NOT oSpeller.CheckSpelling( cString )
   * Something's misspelled, so take action
ENDIF
```

As shown, this class requires VFP 7 because it uses the new GetWordCount() and GetWordNum() functions. However, you can replace them with their FoxTools' equivalents (Words() and WordNum()) to use this code in earlier versions of VFP.

It's also worth noting that the class is written to open Word once and keep it open, closing Word when the class is destroyed. This makes it easy to check the spelling of multiple strings. While the first call may be a little slow because Word is instantiated at that time, after that, it should be pretty quick.

The class library WordUtils.VCX is included on this month's Professional Resource CD.

–Tamar