Word Fields

Part III: Use Events to Push the Envelope

By Cindy Meister

The first two articles in this three-part series on Word fields concentrated on using fields to reduce the amount of code needed to automate the display of information in a document. In this final installment, the emphasis turns to how VBA and Word 2000 events can enhance Word’s field functionality.

Jump to Index Entry by Double-clicking Page Number

In the last few versions of Word, it's been possible to double-click a page number in a Table of Contents in order to jump to that page; since Word 97, including the /h switch in the TOC field has turned each entry into a hyperlink, so that a simple click moves directly to the entry. Similar functionality for indexes is not (yet) included, but is often requested. Word 2000’s WindowBeforeDoubleClick application event offers a solution, as shown in Listing One (beginning on page 36).

As Romke Soldaat’s article “Fighting Desktop Pollution” explained in some detail in the November 1999 issue of *Microsoft Office & VBA Developer*, a class module must be inserted into a project in order to use application events. In it, a public WithEvents variable must be declared. The required event can then be selected from the Procedures list, and the necessary code entered:

```vba
Public WithEvents app As Word.Application
Private Sub app_WindowBeforeDoubleClick(ByVal Sel As Selection, Cancel As Boolean)
    Cancel = False
    Call FindIndexEntry(Sel, Cancel)
End Sub
```

The WindowBeforeDoubleClick event supplies two parameters: Sel, the place where the double mouse-click sets the insertion point in the document; and Cancel, which determines whether the default double-click action will be executed. By default, this value is False; in other words, the default double-click action (selecting a word in the text) will take place. Because the default value may not be known to the person reading the code, or might possibly be changed in a future version of Word VBA, the first line of the procedure sets Cancel explicitly to False.

Control is then passed to a routine in a regular module, with the two parameters as arguments. The value of Cancel will be changed to True if a number was double-clicked in an Index field with a text entry in the same paragraph. In that case, the first instance of text in the document that matches the index entry, on or following the page of the number that was double-clicked, will be selected.

Application events must be explicitly activated and deactivated from within a regular module. First, a variable must be declared as a new instance of the class containing the events. Then it must be set to the application to activate the events, or set to Nothing to turn them off:

```vba
Dim IndexDoubleClick As New clsIndex
Public Sub IndexDoubleClick_Activate()
    Set IndexDoubleClick.app = Word.Application
End Sub
Public Sub IndexDoubleClick_Deactivate()
    Set IndexDoubleClick.app = Nothing
End Sub
```
Bon Mot

You could assign these two procedures to CommandBar controls, or call them from an AutoOpen macro in a document or template, or from an AutoNew macro in a template. AutoOpen executes whenever the container document, or a document attached to the template, is opened. The AutoNew macro executes whenever a new document is created from the template. Keep in mind, however, that the events will be executing in all open document windows, because application events are global.

Now that the structure is ready, it’s time to look at how FindIndexEntry does its job, which can be broken into the following tasks:
1) Discover whether a number was double-clicked.
2) Is so, did the double-click take place in a field?
3) Was it the right type of field (an index field)?
4) If the answer to both questions is yes, then the index entry text must be extracted.
5) Jump to the page number corresponding to the number that was double-clicked, then to the first text that matches the index entry.

The function GetSelNumber returns the entire word at the point where the double-click was made. Keep in mind, this is a point and contains only a single character; a page number may contain up to three or four digits:

```vba
Public Function GetSelNumber(_
    ByVal rng As Word.Range) As String
    ' Extend range to include entire word.
    rng.MoveStart wdParagraph, -1
    ' Exclude the page number.
    rng.MoveEnd wdCharacter, Nz CharPageNr * -1
    CleanedString = rng.Text
    CleanedString = Mid(CleanedString, InStr(CleanedString, vFieldFeed) + 1)
    CleanedString = CleanedString.Trim(CleanedString)
    If InStr(CleanedString, vTab) <> 0 Then
        CleanedString = Left(CleanedString, InStr(CleanedString, vTab) - 1)
    End If
    GetIndexEntry = CleanedString
End Function
```

Of course, the word at the double-click range may not contain only numeric characters. In this case, the subsequent IsNumeric check will return False, and the function will exit returning the value False.

The next step is to determine whether the double-click point was within a field. Interestingly, Word VBA does not include a function that returns this information, although Word itself recognizes when a right-click is performed in a field by displaying Toggle Field Codes and Update Fields in the context menu. The function IsSelectionInField, shown in FIGURE 1, takes advantage of the fact that if the field codes are displayed while the selection is in a field:
1) The insertion point will be just before the closing brace which represents the end of the field (ASCII 21); and
2) Expanding the selection by a word at this point selects the entire field.

Although VB doesn’t have a function to check whether an object variable contains a valid object, Word VBA does: IsObjectValid. Thus, no error-trapping is required to determine whether content of the fld object variable returned by IsSelectionInField is a Word field, or Nothing. If fld is valid and of the type wdFieldIndex (a built-in constant), the code proceeds to get the text of the index entry. Because every index entry is in its own paragraph, getting the text is not difficult, as shown in FIGURE 2. The tricky part is cutting of all the extraneous material, such as trailing spaces and tabs.

As with the WindowBeforeDoubleClick event, the WindowSelectionChange event must be defined in a class module. It calls the UpdateTableFields procedure, passing the index entry text from the surrounding paragraph. At last, GoToIndexEntry jumps to the top of the page whose number was double-clicked, then uses the Find object to select the first instance of the index entry text after this point.

FIGURE 1 (Top): Once you know the insertion point directly precedes the field's closing brace when the field codes are displayed, determining whether the current selection is in a field (and getting the field) is simple. FIGURE 2 (Bottom): Extract the index entry text from the surrounding paragraph.

**Update Table Fields Using Selection Change**

Another bit of functionality missing in all versions of Word is the ability to have calculations in fields (especially in tables) update automatically. The WindowSelectionChange event offers a solution, as shown in Listing Two (on page 38). WindowSelectionChange fires whenever the mouse is used to change the insertion point position, the arrow keys are used, or [Tab] is pressed while in a table. Pressing [Delete] also triggers the event; pressing [Backspace] does not.

As with the WindowBeforeDoubleClick event, the WindowSelectionChange event must be defined in a class module. It calls the UpdateTableFields procedure, passing
First, if the user selects a table cell, row, or column when the Fields.Update command executes, this selection is lost and the insertion point ends up at the beginning of a cell (often the last cell in the current row). After a great deal of experimentation, it was found that the combination of Range and Selection objects successfully updates the fields and restores the selection (see FIGURE 3).

The next problem that becomes evident when actually working with the event is that it is impossible to edit field codes in Word 2000. As soon as a field is updated, the field codes are hidden. Additionally, because every change in selection updates the field codes, there's no way to move into them.

Obviously, the event must be deactivated whenever the user needs to edit field codes, but requiring the user to execute a special command whenever field codes need to be edited is less than ideal. Assuming you want to allow the user to edit field codes! It would be much better to toggle the field codes and deactivate the event in one step, and re-activate the event automatically when the field codes are re-hidden.

The procedure in FIGURE 4, when mapped to a keyboard shortcut that toggles the field codes, calls CalculationUpdate_Deactivate if the class object o_CalculationUpdate.App is present. The False parameter means the keyboard assignment [ALT+P] (Toggle Field Codes) won't be changed. If the class object isn't present, the event is re-activated — again, without affecting the keyboard assignment.

Because AltViewFieldCodes should only execute when the user is working with the automatic field updates, the keyboard should be remapped only when the event is deliberately activated or deactivated. The two procedures in FIGURE 5 take care of this. The ActiveFieldEdit parameter is optional, so the event can also be activated and not allow the user to edit any table fields.

The StartUpdate and EndUpdate procedures call the two event handlers in FIGURE 5. StartUpdate also makes sure that all field codes are hidden in all open document windows so the procedure doesn't execute “backwards.”

Interestingly enough, a new problem appeared when ViewFieldCodes was remapped to AltViewFieldCodes: Every time the fields in a table were updated, the field codes were toggled (Selection.Tables(i).Range.Fields.Update). This is certainly a bug, but easily remedied; simply insert the following line just before rngIsSelected.Select in FIGURE 3:

```
ActiveWindow.View.ShowFieldCodes = False
```

One minor annoyance couldn't be eradicated. When the WindowSelectionChange event executes, it causes the screen to flicker and the mouse pointer to cycle between two or three pointer shapes. The API functions to lock the window or hide the mouse pointer had no noticeable effect on the problem, which is most apparent when holding down an arrow key to move the insertion point multiple times in succession.
Conclusion
This excursion into how Word's new events can enhance the capabilities of Word fields concludes this series. Besides offering information, the articles in this series were written with the intention of stimulating the reader to look beyond the everyday boundaries of an application. A synergy of outlook and ideas often leads to the development of new realizations and solutions. I'm looking forward to seeing what tools and workarounds appear in the future based on Word fields.

The files referenced in this article are available for download at http://www.OfficeVB.com/download/. File name: vba200006cm_f.zip.

Cindy Meister has her own consulting business, INTER-Solutions, based in Switzerland. Prior to becoming independent, she spent three years as a consultant/trainer for Deloitte & Touche in Zurich. Five years experience as head of administration in an international organization and a Bachelor of Science degree provide her with a broad background for understanding customer requirements. Cindy's fully bilingual (English and German), with customers in four countries (and counting) and is a regular contributor to the German edition of Inside Word. Four years as a Microsoft MVP for MSWord support and as Sysop in the CompuServe MSWord forum have given her in-depth knowledge of Microsoft Office and Word. For general questions on Word and links to other useful sites, visit her Web site at http://homepage.swissonline.ch/ cindymeister. You can reach her at cindymeister@swissonline.ch.

Begin Listing One — WindowBeforeDoubleClick
· clsIndex, a class module.
  Option Explicit

Public WithEvents app As Word.Application

Private Sub app_WindowBeforeDoubleClick(_
  ByVal Sel As Selection, Cancel As Boolean)
  Cancel = False
  Call FindIndexEntry(Sel, Cancel)
  End Sub

  ' nIndex, a regular module.
  Option Explicit

Dim IndexDoubleClick As New clsIndex

Public Sub IndexDoubleClick_Activate()
  Set IndexDoubleClick.app = Word.Application
  End Sub

Public Sub IndexDoubleClick_Deactivate()
  Set IndexDoubleClick.app = Nothing
  End Sub

  ' If a number was double-clicked in an Index field, find the first instance of the index entry in the document text, starting from that page number. Returns True if a number was double-clicked in an Index field and there is text (an entry) in the same paragraph.
  Public Function FindIndexEntry(_
    ByVal Sel As Word.Selection, _
    ByVal Cancel As Boolean) As Boolean

  Dim SelectedText As String
  Dim rngSel As Word.Range
  Dim FindString As String

  FindIndexEntry = Cancel
  Application.ScreenUpdating = False
  ' Define a range object on the range of the IP at the time of the double-click.
  Set rngSel = Sel.Range
  ' Get all consecutive numbers in the range.
  SelectedText = GetSelNumber(rngSel)
  ' If range based on selection is not numeric, stop.
  If IsNumeric(SelectedText) = False Then
    Exit Function
  End If

  ' Set object to field in selection.
  Set fid = IsSelectionInField(Sel)
  ' If no field, fid is nothing and nothing happens.
  If IsObjectValid(fid) Then
    ' Else check if it's an index field.
    If fid.Type = wdFieldIndex Then
      ' Get the index entry.
      FindString = GetIndexEntry(rngSel)
      ' If there is an entry.
      If Len(FindString) > 0 Then
        ' Turn off default double-click behavior.
        FindIndexEntry = True
        ' Jump to entry match in text.
        If there's no match in text, will go to XE field; if XE fields are hidden, will select entry in the index.
        Call GoToIndexEntry(SelectedText, FindString)
      End If
    End If
  End If
  End If
  Application.ScreenUpdating = True
  End Function

Public Function GetSelNumber(_
    ByVal rng As Word.Range) As String

    ' Extend range to include entire word.
    rng.Expand wdWord
    GetSelNumber = rng.Text
    End Function

    ' Returns the field containing the select.
    ' If no field, then function returns 'Nothing'.
    Public Function IsSelectionInField(_
    ByVal Sel As Word.Selection) As Word.Field

    ActiveWindow.View.ShowFieldCodes = True
    ' Check if selection is at end of field
    If Asc(Sel.Characters(1)) = 13 Then
      ' Extend the selection to entire field.
      Sel.Expand wdWord
      Set IsSelectionInField = Sel.Fields(1)
    Else
      Set IsSelectionInField = Nothing
    End If
    ActiveWindow.View.ShowFieldCodes = False
    End Function

    ' Returns text at beginning of paragraph for specified range. All trailing spaces, tabs, non-printing characters, etc. are cut off.
    Public Function GetIndexEntry(_
    ByVal rng As Word.Range) As String

    Dim CleanedString As String
    Dim NrCharsPageNr As Long
    NrCharsPageNr = Len(rng)
    ' Extend range to include entire entry.
    rng.MoveStart wdParagraph, _
    ' Excluding the page number.
    rng.MoveEnd wdParagraph, NrCharsPageNr - 1
    ' Clean all spaces, forseems and tabs
    ' to get Index entry.
    CleanedString = rng.Text
    CleanedString = Replace(CleanedString, _
      InStr(CleanedString, vbCrLf) + 1, _
      CleanedString = CleanString(Text.InStr(CleanedString)) _
      If InStr(CleanedString, vbCrLf) <> 0 Then
        CleanedString = Left(CleanedString, _
          InStr(CleanedString, vbCrLf) - 1)
          JUNE 2000
End If
GetIndexEntry = CleanedString
End Function

Private Sub DoToIndexEntry(ByVal Pgnr As String, _
ByVal FindString As String)
' Go to page of index entry.
Selection.DoTo What:=wdGoToPage, _
Which:=wdGoToAbsolute, Count:=Val(Pgnr)
' Find the first entry on this or a following page.
With Selection.Find
.ClearAllFuzzyOptions
.ClearFormatting
.Text = FindString
.Forward = True
.Wrap = wdLinedAsk
.Execute
End With
End Sub

End Listing One

Begin Listing Two — WindowSelectionChange
Option Explicit
' clsTables

Public WithEvents App As Word.Application
Private Sub App_WindowSelectionChange(ByVal Sel As Selection)
    UpdateTableFields Sel
End Sub

Option Explicit
Dim oCalculationUpdate As New clsTables

Private Const ERR_DOCVARNOTPRESENT = 5825
Private Const ERR_INVALIDRANGE = 4608

Sub StartUpdate()
    Dim wn As Word.Window
    ' Hide field codes in all windows.
    For Each wn In Application.Windows
        wn.View.ShowFieldCodes = False
    Next wn
    ' Before turning on automatic update
    ' with re-routing of ViewFieldCodes.
    CalculationUpdate_Activate True
End Sub

Sub EndUpdate()
    ' Turn off automatic field update
    ' and reroute ViewFieldCodes to default.
    CalculationUpdate_Deactivate True
End Sub

Sub CalculationUpdate_Activate( _
Optional ActiveFieldEdit As Boolean)
    ' Activate automatic field updates in tables.
    Set oCalculationUpdate.App = Word.Application
    Remap Alt+F9 (ViewFieldCodes) so automatic updating is
    turned off when field codes are toggled on to allow
    editing of fields in tables.
    If Not IsMissing(ActiveFieldEdit) Then
        If ActiveFieldEdit = True Then
            CustomizationContext = NormalTemplate
            KeyBindings.Add wdKeyCategoryMacro,
            'AltViewFieldCodes', wdKeyAlt + wdKeyF9
        End If
        End If
    End Sub

Sub CalculationUpdate_Deactivate( _
Optional ActiveFieldEdit As Boolean)
    ' Turn off automatic field updating in tables.
    Set oCalculationUpdate.App = Nothing
    Remap Alt+F9 to default.
End Sub

If Not IsMissing(ActiveFieldEdit) Then
    If ActiveFieldEdit = True Then
        CustomizationContext = NormalTemplate
        FindKey(BuildKeyCode(wdKeyAlt, wdKeyF9)).Clear
    End If
End If
End Sub

Sub UpdateTableFields(ByVal Sel As Word.Range)
    Dim rngWasSelected As Word.Range
    Dim rngISelected As Word.Range
    On Error GoTo Err_UpdateTableFields
    With ActiveDocument
        ' Note previous and current selection points.
        Variables("WasSelected") = Variables("IsSelected")
        Variables("IsSelected") = Sel.Range.Start
        Set current ranges.
        Set rngWasSelected = Selection.Range
        ' If selection is in a table
        ' update all fields in the table.
        If Sel.Information(wdWithinTable) Then
            Selection.Tables(1).Range.Fields.Update
            ' Compensate for bug: rehide field codes.
            ActiveWindow.View.ShowFieldCodes = False
            ' Selection is lost when fields are updated.
            ' so reselect the range as the user had it.
            rngISelected.Select
        Exit Sub
    End If
    ' Set last selection ranges.
    Set rngWasSelected = .Range(Start:=Variables("WasSelected"),Value:=1)
    End:=Variables("WasSelected"),Value:=1)
    ' If the selection just left the table
    ' update all the fields in that table.
    If rngWasSelected.Information(wdWithinTable) And _
        rngISelected.Information(wdWithinTable) = False
        rngWasSelected.Tables(1).Range.Fields.Update
    End If
End Sub
End Sub

Err_UpdateTableFields:
Select Case Err.Number
Case ERR_DOCVARNOTPRESENT
    ' Document variable not yet defined.
    ' Create the two required by this procedure.
    ActiveDocument.Variables.Add _
    Name="IsSelected", Value:=1
    ActiveDocument.Variables.Add _
    Name="WasSelected", Value:=1
    Set rngWasSelected = _
    ActiveDocument.Range(Start:=1, End:=f1)
    Resume Next
Case ERR_INVALIDRANGE
    ' Character number larger than available
    ' number of characters in doc.
    Set rngWasSelected = ActiveDocument.Range( _
    Start:=ActiveDocument.Range.End - 1, _
    End:=ActiveDocument.Range.End)
    Resume Next
Case Else
    MsgBox 'Error number: ' & Err.Number & vbCr & _
    'Message: ' & Err.Description
End Select
End Sub

Sub AltViewFieldCodes()
    ' Alternative ViewFieldCodes (Alt+F9) that turns off
    ' automatic field updating to allow editing of field
    ' codes in tables.
    If IsObjectValid(oCalculationUpdate.Activate) = True Then
        CalculationUpdate_Deactivate False
    Else
        CalculationUpdate_Activate False
    End If
    ActiveWindow.View.ShowFieldCodes = _
    Not ActiveWindow.View.ShowFieldCodes
End Sub