Build Custom App with Word, Outlook

by Chris Barlow

Use Word 97, Outlook, and VB5 to automate your holiday thank-you notes.

Click & Retrieve Source CODE!

It is holiday season again—can you hear your mother asking, "Have you written your thank-you notes?" When my children were young, they quickly learned the word processor's mail-merge function was a convenient way to create thank-you notes. Now that Microsoft Word includes Visual Basic, and Microsoft Outlook's objects are accessible through Visual Basic, Microsoft Office gives you the ideal platform to keep track of gifts you receive this holiday season and generate thank-you notes.

This month, I'll show you how to create a small VB app that runs within Word and executes a mail-merge based on data it gathers from Outlook. I'll be the first to admit there are easier ways to do this—but this will be a fun way to learn how to use Outlook's custom fields and get a better understanding of Word's programmability.

One of the neatest things about Outlook is the way it exposes the MAPI information store. The items in your Inbox appear as e-mail messages. But when you change to your Contacts folder, they appear as contacts with names and addresses. These different items are all stored in the same file. The Outlook object model exposes the items in the different folders as Appointment, Contact, Journal, Mail, Note, or Task items. Each of these items has different properties and methods. In addition, *and this is the cool part*, you can add your own custom properties to any of the folders or individual items. You can view and search these custom properties like the intrinsic properties.

It's hard to design a powerful Personal Information Manager (PIM). Users always find a feature critical to them missing in their PIM. Most vendors predefine a set of User fields in their database and forms and let the users enter data in these fields, but users always run out of these fields. Outlook, on the other hand, makes it easy to add

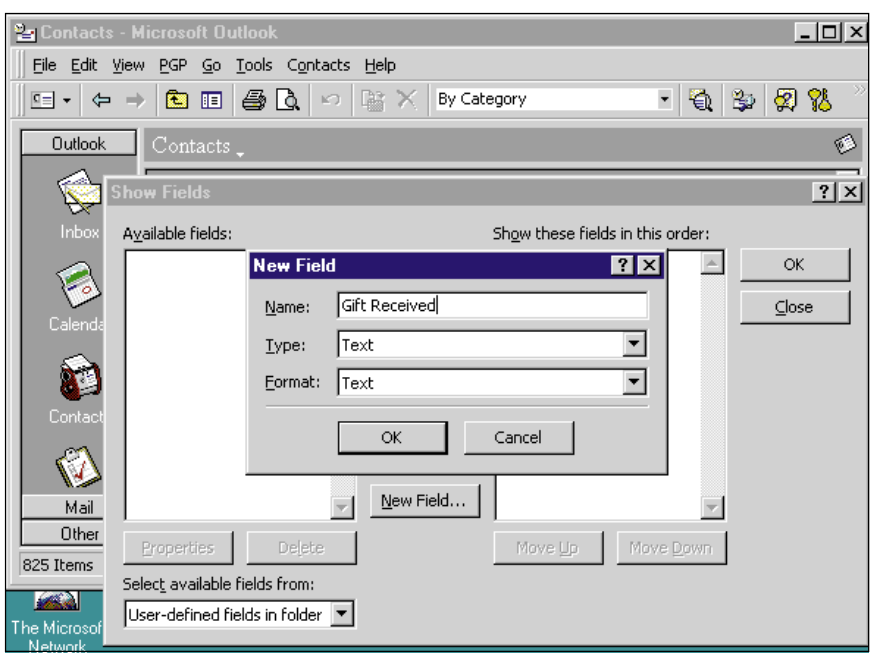

FIGURE 1 ***Create a New Field.*** *I love this feature! If you need to keep track of special information, you can create a new field and add it to a view.*

Chris Barlow is president and CEO of SunOpTech, a developer of manufacturing decision-support and supply chain applications. He's also a frequent speaker at VBITS, Tech•Ed, and DevDays, and has been featured in two Microsoft videos. Reach Chris at ChrisB@SunOpTech.com or through SunOpTech's Web server at www.SunOpTech.com.

and view custom properties. You can define as many as you need from a selection of 11 different data types. You're never going to run out.

The first step in building your application is to add a custom property to your Contacts folder called "Gifts Received." As you receive a gift during the holiday season, use a custom view to enter the gift in this field.

Start Outlook and view your Contacts folder. Click on the View menu, and select the Define Views menu item to display a list of your current views for this folder. Click on the New button, name the new view "Gifts," and use a Table

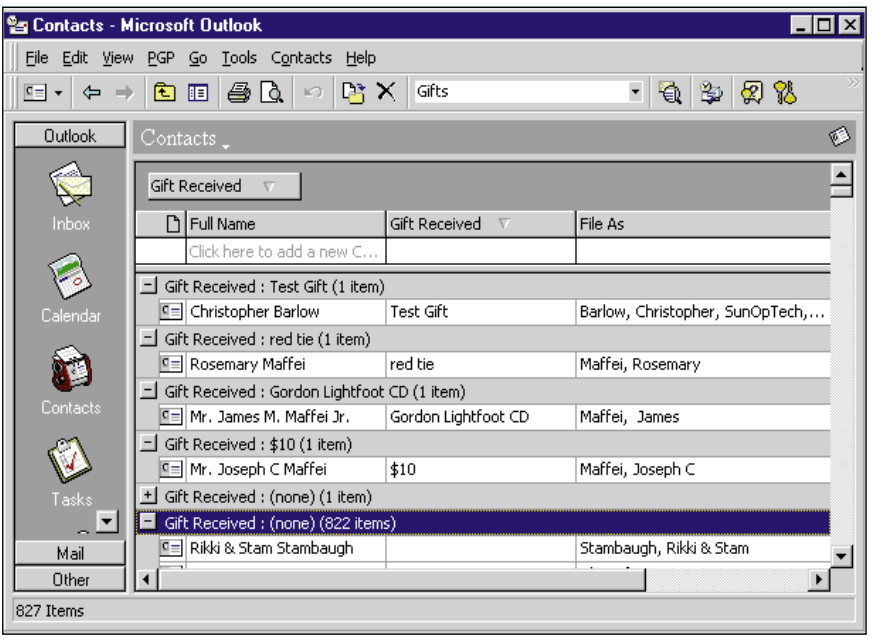

FIGURE 2 ***The New Gifts View.*** *The newly created view with the Gift Received field that you added looks just like any other view in Outlook.*

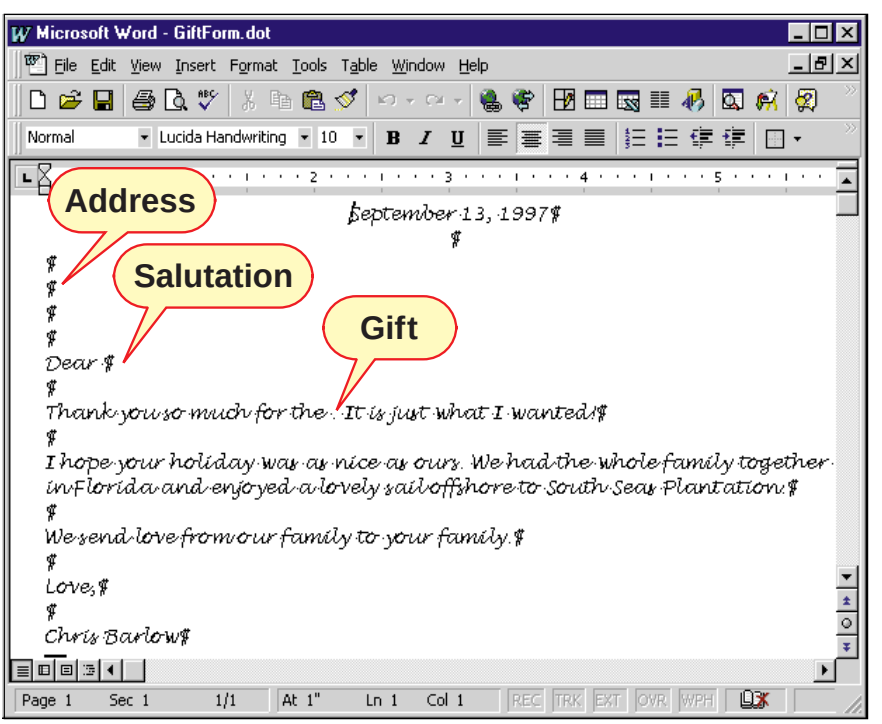

FIGURE 3 ***The Thank-You Note Template.*** *When you save your document as a template, Word can create new documents based on the template. The new document contains the special bookmarks you have created to be replaced by your Visual Basic procedure.*

view. Outlook generates a simple table view with default fields. Click on the Fields button to display the available contact fields, and click on the New button to create a field called Gift Received (see Figure 1). Save your new view and display the Contacts folder using your new Gifts view (see Figure 2). Enter a few gifts next to some of the contacts for testing purposes, and you're ready to start on the Visual Basic procedures.

Now that Visual Basic is built into so many applications, you can choose the application to host your Visual Basic code. You don't need to change the code to fit the host because each app uses the same Visual Basic engine. This application will use Word's mail-merge functionality, so you'll write the code with a Word document.

First, plan the functionality of your application. Your app will establish a connection to Outlook and select only the contacts who sent you a gift. Then it will load the name, address, and gift information from those contacts into a merge format acceptable to Word and execute a mail-merge using a generic thank-you note that you have prepared.

Start Word and prepare the template for the thank-you notes. Type the text for your thank-you note, and insert three bookmarks named "Address," "Salutation," and "Gift" where the application substitutes the fields from your contacts. Save the template as "GiftForm.Dot" (see Figure 3).

Press Alt-F11 to launch Word's Visual Basic editor. Look familiar? This is nearly the same development environment you're accustomed to in VB5. Unlike VB5, however, your code won't be saved as separate BAS and CLS files. Instead, you can save your code either as a template or a document. You can download the entire code listing for this article on the free, Registered Level of DevX (see the Code Online box for details).

Click on the References menu item on the Tools menu, and add Microsoft Outlook Libraries as a reference. Insert a module into the document, and enter this code that follows the functional outline. The first three lines dimension global variables for the connection to Outlook, the Contacts folder, and the contact items. The CreateThankYou procedure simply calls the three procedures you will write:

```
Dim ol As New Outlook.Application
```

VB5

```
For Row = 1 To GiftItems.Count
  With GiftItems(Row)
     'build data in doc
     adoc.Tables(1).Cell(Row + 1, 1).Range.Text = _
        .FirstName
     adoc.Tables(1).Cell(Row + 1, 2).Range.Text = _
        .LastName
     adoc.Tables(1).Cell(Row + 1, 3).Range.Text = _
        .MailingAddressStreet
     adoc.Tables(1).Cell(Row + 1, 4).Range.Text = _
        .MailingAddressCity
     adoc.Tables(1).Cell(Row + 1, 5).Range.Text = _
        .MailingAddressState
     adoc.Tables(1).Cell(Row + 1, 6).Range.Text = _
        .MailingAddressPostalCode
     If Left(.UserProperties![Gift Received].Value, _
        1) = "$" Then
        adoc.Tables(1).Cell(Row + 1, 7).Range.Text = _
           "generous gift"
     Else
        adoc.Tables(1).Cell(Row + 1, 7).Range.Text = _
           .UserProperties![Gift Received].Value
     End If
  End With
Next
adoc.SaveAs "giftmerg.doc"
End Sub
```

LISTING 1 ***Move Data from Outlook.*** *This routine loops through the selected contacts from Outlook and moves the data to a table in a Word document to act as the data source for the mail-merge.*

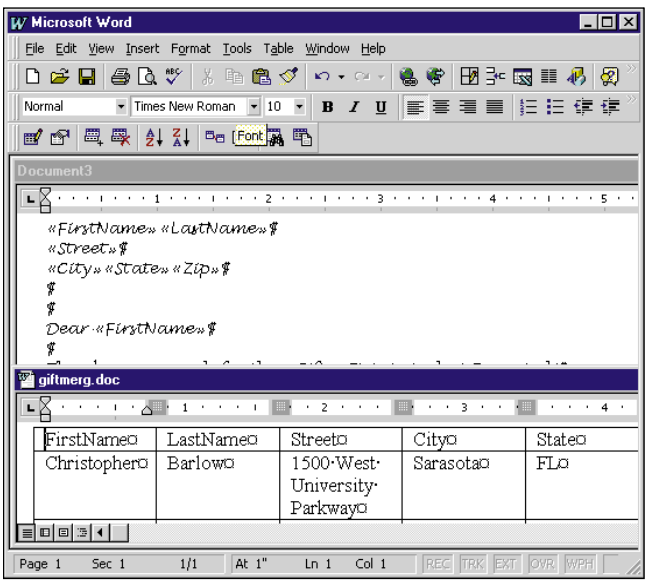

FIGURE 4 ***Mail-Merge Files.*** *This figure shows both the mail-merge data file with the table created by the Visual Basic application, as well as the main document with the embedded mail-merge fields.*

VB5

```
Sub DoMailMerge()
Documents.Add "giftform.dot"
ActiveDocument.MailMerge.OpenDataSource _
  Name:="giftmerg.doc"
ActiveDocument.MailMerge.EditMainDocument
Selection.GoTo What:=wdGoToBookmark, Name:="Address"
With ActiveDocument.MailMerge.Fields
  .Add Range:=Selection.Range, Name:="FirstName"
  Selection.InsertAfter " "
  .Add Range:=Selection.Range, Name:="LastName"
  Selection.InsertParagraphAfter
  Selection.MoveDown wdLine
  .Add Range:=Selection.Range, Name:="Street"
  Selection.InsertParagraphAfter
  Selection.MoveDown wdLine
  .Add Range:=Selection.Range, Name:="City"
  Selection.InsertAfter ", "
  .Add Range:=Selection.Range, Name:="State"
  Selection.InsertAfter "  "
  .Add Range:=Selection.Range, Name:="Zip"
  Selection.GoTo What:=wdGoToBookmark, _
     Name:="Salutation"
  .Add Range:=Selection.Range, Name:="FirstName"
  Selection.GoTo What:=wdGoToBookmark, Name:="Gift"
  .Add Range:=Selection.Range, Name:="Gift"
End With
With ActiveDocument.MailMerge
  .Destination = wdSendToNewDocument
  .Execute
End With
End Sub
```

LISTING 2 ***Create the Mail-Merge Document.*** *This routine replaces the bookmarks in the thank-you note template with the merge field names in the data source document.*

```
Dim ContactFolder As MAPIFolder
Dim GiftItems As Items

Sub CreateThankYou()
LoadContacts
CreateDataDoc
DoMailMerge
End Sub
```

The LoadContacts procedure makes a connection to the Contacts folder within Outlook's MAPI namespace and uses the Restrict method of the Items collection to select only those contacts with an entry in the Gift Received field:

```
Sub LoadContacts()
Set ContactFolder = _
   ol.GetNamespace("MAPI").
   GetDefaultFolder (olFolderContacts)
Set GiftItems = ContactFolder.Items._
   Restrict("[Gift Received] > ''")
End Sub
```

The data source for Word's mail-merge must be in the form of columns and rows. Word can create a data source from tables in Word documents, ranges in Excel workbooks, ASCII files, ODBC data sources, and MAPI address books. Because your Contacts folder is a MAPI address book, you might think you can use it directly. However, the custom properties, such as Gift Received, are not available.

It's easier to create a new document and add a table with seven columns for the required fields and the field names in the first row. You know how many contacts have been selected by the Count property of the GiftItems collection:

```
Sub CreateDataDoc()
Dim Row&
Dim adoc As Document
Set adoc = Documents.Add
With adoc
   .Tables.Add Range:=adoc.Content, _
      NumRows:=GiftItems.Count + 1, _
      NumColumns:=7
   .Tables(1).Cell(1, 1).Range.Text = _
      "FirstName"
   .Tables(1).Cell(1, 2).Range.Text = _
      "LastName"
   .Tables(1).Cell(1, 3).Range.Text = _
      "Street"
   .Tables(1).Cell(1, 4).Range.Text = _
      "City"
   .Tables(1).Cell(1, 5).Range.Text = _
      "State"
   .Tables(1).Cell(1, 6).Range.Text = _
      "Zip"
   .Tables(1).Cell(1, 7).Range.Text = _
      "Gift"
End With
```

Now continue to build the merge table by looping through the GiftItems collection and setting the contact data into the proper row. Notice that you reference the custom property by using the default Item method of the UserProperties collection—*UserProperties![Gift Received].Value*. The exclamation point delimits a text field name. You could write this as *UserProperties("Gift Received").Value*. Note that there is one change to the gift data contained in the contact record—my wife was taught that you never mention the dollar amount of a cash gift, but say "generous gift" instead. The If statement makes this substitution (see Listing 1).

Finally, the application creates a new document based on the thank-you note template that you saved. Then search for the bookmarks to insert the mail-merge fields and execute the actual mail-merge (see Listing 2 and Figure 4).

Resize the Visual Basic window so you can see the Word window, and select the Macros menu item from Word's Tools menu. Click on the Step Into button so you can single-step through your code. As you single-step, it's fun to watch your procedure build the mail-merge table and execute the mail-merge. You can see how easy it is to drive Outlook and Word with Visual Basic.

Code Online

You can find all the code published in this issue of VBPJ on The Development Exchange (DevX) at http://www.windx.com. For details, please see "Get Extra Code in DevX's Premier Club" in Letters to the Editor.

Build Custom App with Word, Outlook

Locator+ Codes

Listings ZIP file plus the entire code listing related to the article (free Registered Level): VBPJ1297

Listings for this article plus the entire code listing, as well as VB code in a Word document used to generate thank-you notes, and the template that contains the copy for the thank-you note and the bookmarks to be replaced by the mail-merge fields (subscriber Premier Level): GS1297P