Making contact

To Outlook, everything is a message. But how do you get that message across? Tim Anderson explains.

The Outlook contact manager is a handy way to store addresses, with strong features like integration with the Outlook calendar and synchronisation with Windows CE palmtops. Integration with the rest of Office, though, is poor. A common question is how to use user-defined fields in a Word mail-merge operation. I don't know of any way to do so without custom programming.

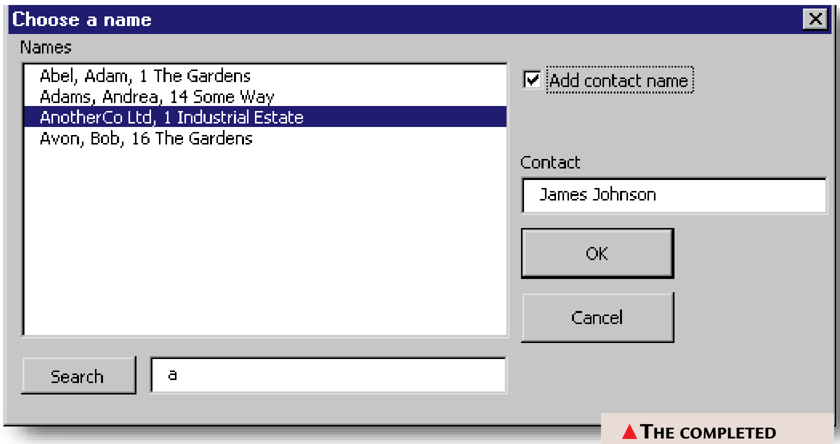
The next snag is that Outlook is harder to program than the rest of Office. The problem is not so much the use of VB Script rather than VBA, but Outlook's strange object model along with some surprising design features. The problem seems to be that Outlook's origins as a mail client have unduly influenced the Personal Information Manager features. To Outlook, everything is a message.

I recently completed a solution that uses Outlook contacts as a data source for Word, with access to user-defined fields. The basic requirement was for a VBA macro in Word to show a dialogue letting you search the Outlook contacts folder. Pick a contact, and the macro will insert field values into a Word document. We can't show the whole solution here, but the following tips give you all you need to know.

How do you use a custom form?

To design an Outlook custom form, open a contact (for example) and choose Tools, Forms, Design this form. There is a control toolbox and a field chooser, and you can include ActiveX controls by right-clicking the toolbox and picking the Custom control option. Form, View code opens the very basic script editor.

The tricky bit is how to use the form. If you simply save your customised form, it is saved only with the specific contact you have been working on. To have the Contact folder use your form as the default, take two further steps. With the form open, choose Tools, Forms, Publish form. Unless you are running Exchange Server, the natural home for custom forms is the Personal Forms Library, available from the combo box in the Publish Form dialogue.


Choose a name

Names

Abel, Adam, 1 The Gardens
Adams, Andrea, 14 Some Way
AnotherCo Ltd, 1 Industrial Estate
Avon, Bob, 16 The Gardens

☑ Add contact name

Contact: James Johnson

OK

Cancel

Search: a

▲The completed solution in Word searches matching names and retrieves additional information for a selected contact

Next, close the form and go to the folder list. Choose View, Folder List if it is not already showing. Right-click the Contacts folder and choose Properties. Drop down the "When posting to this folder" combo box and choose Forms. Find your custom form. When you select it, look at the bottom of the dialogue and note the Message Class. This will be something like "IPM.Contact. MyContact", depending on the name you chose when publishing the form. Note this for future reference.

Now you will find your custom form is used when you create a new contact. Unfortunately, any existing contacts still use the old form. This is because the form used by a contact is determined by its MessageClass property, which you can only access programmatically. Fig 1 [p294] shows a possible script, which you can run from a button on an Outlook form.

A handy tip is to create a new contact and call it Utility. Use this to store house-keeping scripts such as one that updates the MessageClass. You can also use it to store global variables. You need to publish the Utility form, even though it is used by only one item, to prevent warning messages about dangerous macros.

What is the unique identifier for a contact?

Trick question. As far as I can tell, there is no such thing. Internally there probably is, but it cannot be accessed. A data source without a unique identifier for each record is a hopeless case, so the answer is to create one. There are many ways to do this, but the approach I took was to create a custom property in the utility contact, using the New button on the All fields tab. I called this MaxID. It stores a number representing the last used ID number for a contact.

The next step is to use a custom form for new contacts in the folder. Add an event handler for the Open event. Fig 2 [p294] shows code that reads the last-used ID, increments it by one, and uses the result for the new ID. I chose to store the ID in the built-in Customer ID field, but you could also use a user-defined field. Be sure to set it to 0 in the form you publish, otherwise the code in Fig 2 will not work.

If this is a multi-user system using Net folders, you could append the username to the ID. Otherwise there is a risk of duplicates. Another possibility would be to query an external database. This would let you link an Outlook contact with an Access or SQL Server record, the beginnings of a true integrated system.

Fig 2 uses the Typename function to discover if the utility contact has been found. The contact is a COM object, and if the Find method fails, the contact variable will have an invalid reference. Typename is able to test its validity without raising an error in the process.

[FIG 1] Changing the MessageClass

```
Sub CmdUpdate_Click()
Set ns = application.Getnamespace("MAPI")
Set flder = ns.Folders("Personal Folders")
Set cts = flder.Folders("Contacts").items

If ct.FileAs  "Utility" Then
' don't update utility form
ct.MessageClass = "IPM.Contact.MyContact"
ct.save
End If
Next

msgbox "All done"

End Sub
```

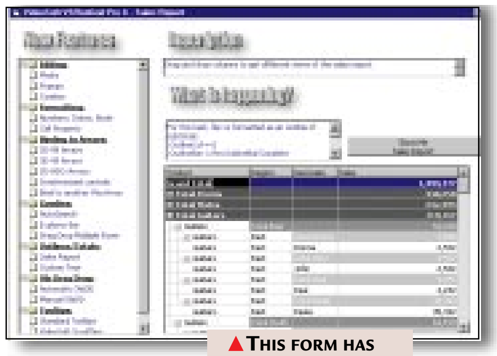
[FIG2] Adding a unique ID

```
Function Item_Open()

If item.CustomerID = 0 then
Set cts = item.parent.items
Set utilitem = contacts.find("[File As]='Utility'")

If Typename(utilitem) = "Nothing" then
Msgbox "Error finding utility item"
Else
Set utilfld = utilitem.userproperties.find("MaxID")
NewID = utilfld.value + 1
item.CustomerID = NewID
utilfld.value = NewID
utilitem.save
End If

End If

End Function
```

How do you search contacts from Word?
Outlook's object model is accessible from Word. The starting point is to open Word's VBA editor, choose Tools, References, and check the Outlook 98 Type Library. Fig 3 [p296] shows how you can get a reference to the Contacts folder, sort it, and search for an individual contact. Sorting is optional, but important if you want to display a list of matches.

Fig 3 also shows how you can retrieve the value of a user-defined field, by means of the Find method of each contact's UserProperties collection.
The ID is important because it lets you put search results in a list box, using the ID to retrieve the chosen contact.
If you have used VBA in Word, you will find it easy to extend this by placing field values at bookmarked points in a Word template, for example.

A more flexible grid
Nice product, shame about the grid. This was a common VB complaint in its early days, and although the FlexGrid in Visual Basic 6.0 improves on previous efforts, there is a lot more you can do with a grid. Doubters need look no further than VideoSoft's FlexGrid Pro. Supplied on one modest floppy disk together with a printed manual, this is the grown-up version of the FlexGrid bundled with VB. It can be bound to either DAO (Data Access Objects) or new-style OLE DB data sources, with read-write access to the data and mask editing available.


▲ THIS FORM HAS TWO FLEXGRID CONTROLS, BOTH USING THE OUTLINE FEATURES OF VSFLEXGRID PRO

VsFlexGrid has many features. It is an outline control as well as a grid, or possibly a hybrid grid-outline, since you can easily create outline grids that hide and expand groups of rows. The Explorer bar lets you click on a heading to sort by that column, and autosearch provides incremental search when the user types on the grid. Autosize columns work as you would expect. ScrollTips show a tooltip as you drag the scroll thumb, revealing the contents of the current row, and an OwnerDraw property lets you write code for the DrawCell event, for total control over what is displayed.

Another nice feature is OLE drag-and-drop, which lets you drag text from the grid to the likes of Word or Excel without using the clipboard. You can also drag-and-drop columns to group data. For applications that make serious use of a grid, VsFlex is recommended.

As a bonus, the VsFlexString component bundled with VsFlexGrid is a pattern-matching control, so that you can search and replace strings using wildcard expressions.

Stringing along
Ed Ross writes: "*I would like to know about API functions: how to access them from Visual Basic, particularly the GetUserName function. I've added the declare from the API viewer.*" Brunel Kirby asks: "*I am having a serious problem with passing strings to and from a Delphi 4 DLL, when using VBA from Word 97 as the calling application. My code is given below.*" [→ code continues on next line.]

```
function StringFunction(P1 : →
Integer) : String; stdcall;
begin
Result :=3D 'Brunel';
end;
```

The problem here is that strings are

[FIG 3]

Code to search Contacts from Word

```
Sub SearchContacts()
Dim ol As Outlook.Application
Dim flder As Outlook.MAPIFolder
Dim cts As Outlook.Items
Dim ct As Outlook.ContactItem

Set ol = CreateObject("Outlook.Application")
Set flder = ol.GetNamespace("MAPI").Folders("Personal →
     Folders")
Set cts = flder.Folders("Contacts").Items
cts.Sort "FileAs", False

'Find a contact
Set ct = cts.Find("[File As] = '" + "Utility" + "'")

If TypeName(ct) = "Nothing" Then
MsgBox "Contact not found"
Exit Sub
Else
MsgBox "Contact name: " + ct.FileAs
MsgBox "Customer ID: " + Str(ct.CustomerID)
MsgBox "Max ID: " + Str(ct.UserProperties.Find →
     ("MaxID").Value)
End If

End Sub
```

[→ code continues on next line]

[FIG 4]

Calling GetUserName from VB

```
Private Declare Function GetUserName Lib →
     "advapi32.dll" Alias "GetUserNameA" →
     (ByVal lpBuffer As String, nSize →
     As Long) As Long
Private Sub cmdShowUser_Click()
Dim sUser As String
Dim lRetVal As Long
Dim lSize As Long

lSize = 255
sUser = String$(lSize + 1, " ")
lRetVal = GetUserName(sUser, lSize)

If lRetVal  0 Then
sUser = Left$(sUser, InStr(1, sUser, Chr(0)))
'could use lSize which also now contains length →
     of string
MsgBox "User name: " + sUser
Else
MsgBox "Function failed"

End If
End Sub
```

[→ code continues on next line]

more awkward to manage than other data types. Since a string is really a set of characters, they are usually managed by means of a pointer variable. This contains the address of the string's starting point. The end of the string is either marked by a special character, or determined by a length parameter stored with the string. The C language uses the former method, called null-terminated strings, while Basic and Delphi use the second approach but in slightly different ways. The Basic string, or BSTR, is also used by COM.

Ed and Brunel both need to return string values from a DLL function called by Visual Basic. The most common way to do this is to declare the string in VB, allocate some space to it, and pass it to the DLL function as a ByVal parameter. This appears to the DLL as a pointer value. This works fine, although you should check for the position of the null terminator otherwise you may find spurious characters in the string. Fig 4 shows an example.

Brunel could use the same technique with Delphi. Instead of returning a string, the function could take a pchar parameter and copy the required value into it [→ *code continues on next line*]:

```
function StringFunction →
(strvar : pchar): longint; →
stdcall;
begin
strcopy(strvar,'Brunel');
Result := 1;
end;
```

This can be called from VB in a similar way to GetUserName. For a safer function, include a MaxLen parameter as well. Then you can allocate MaxLen characters in VB, less one for the null terminator, and use StrLCopy at the Delphi end to ensure that only MaxLen characters are copied. Another possibility is to have a second function that returns the string length.

Finally, Brunel could use the TBstr type, declared in Delphi 4's ActiveX unit, to return a Basic string [→ *code continues on next line*]:

```
function VBStringFunction: →
TBstr; stdcall;
var
sVar: string;
begin
sVar := 'Hello from Delphi';
Result := sysAllocString →
(PWideChar(sVar));
end;
```

PCW CONTACTS

Tim Anderson *welcomes your queries and tips. He can be contacted c/o the PCW editorial office (address, p10) or email* **visual@pcw.co.uk**.

VsFlexGrid Pro costs £199 + VAT (£233.83) from Contemporary Software 01344 873434. Further information and trial version at **www.videosoft.com**.