May, 2001

## Advisor Answers

## Adding custom properties in Outlook

VFP 6.0/7.0, Outlook 2000

Q: Outlook lets me add custom fields to a Contact record (using the All Fields page of the Contact form). How can I do the same thing programmatically? I need to add fields such as AdjusterName, InsuranceCompany, and so forth.

–Bob Barnes (via CompuServe)

A: As you know, Outlook (like the other Office applications) can be used programmatically through Automation. To connect to Outlook, you issue:

```
oOutlook = CreateObject("Outlook.Application")
oNameSpace = oOutlook.GetNameSpace("MAPI")
```

After this, you have access to the entire Outlook object model. (You'll find documentation for that model in the Outlook VBA Help file – VBAOutl9.CHM for Outlook 2000. In addition, Andrew Ross MacNeill has written several articles about automating Outlook. See the June through October, 2000 issues. )

To create a Contact item, you use the CreateItem method of the Outlook application object:

```
#DEFINE olContactItem 2
oContact = oOutlook.CreateItem( olContactItem)
```

To access an existing item, you first need to get a reference to the Contacts folder, then find the one you want:

```
#DEFINE olFolderContacts 10
oContacts = oNameSpace.GetDefaultFolder(olFolderContacts)
oBill = oContacts.Items("Bill Gates")
```

It's worth noting that there's more than one way to find the particular contact item you want. In the example above, I simply specified the full name of the person I wanted. For a Contact object, the full name is also stored in the Subject property, and can be used as an index to the Contacts folders' Items collection. (Each type of item in Outlook has

such a property; for most of them, it's the value that appears in Outlook as the subject.)

The second way to get a particular Contact record is to use the Items collection's Find method. Setting up a Find is a little tricky. The Find method accepts a character string as the search string. The string must be composed of comparisons combined with the AND and OR operators; NOT is also permitted. Each comparison must be in the form:

```
[<Property>] <comparison operator> <constant>
```

For example:

```
[LastName] = "Smith"
```

So, to find a contact record given an email address, we can use code like:

```
cFilter = '[Email1Address] = "' + cEmail + ;
          '" OR [Email2Address] = "' + cEmail + ;
          '" OR [Email3Address] = "' + cEmail + '"'
oContact = oContacts.Items.Find(cFilter)
```

Note that you need to put double-quotes around character strings in the filter and field names must be enclosed in square brackets, so I use single quotes to indicate strings while building the filter.

The Find method is case-insensitive, so you don't need to worry about the case of either the field or the search string. Find doesn't, however, have a setting like VFP's SET EXACT. You can use the <= and >= operator to land on the nearest item (similar to VFP's SET NEAR). Be forewarned, though, that the order of the items in the collection is the order in which they were added. If you want to search them in a particular order, you need to create a separate object reference to the Items collection, sort it on the specified field, then use Find, like this:

```
oContactItems = oContacts.Items
oContactItems.Sort("[LastName]")
cFilter = '[Lastname] = "' + cLastName + '"'
oContactItems.Find( cFilter )
```

What if Find doesn't turn up the right item? Like VFP's LOCATE/CONTINUE, Find has a companion method. FindNext returns the next item that matches the condition specified in the filter. You can keep looking until you find the desired Contact.

You can also find a particular item by looping through the collection and doing comparisons, but this approach is likely to be significantly slower than using Find. In my tests with only 47 items in the collection, depending on the location of the matching item in the collection, looping took anywhere from 4 times to 70 times as long as using Find.

Once you have the Contact you want, whether it's a new one or an existing contact, adding custom properties is easy. Call the Add method of the UserProperties collection. This method requires two parameters: the name for the new property and its data type. The type comes from the olUserPropertyType group. For example, olText is 1, olNumber is 3 and olDateTime is 5. Don't forget that to use these constants in VFP code, you have to #DEFINE them first. So, to add a property to hold the name of the Insurance Company, you use code like:

```
#DEFINE olText 1
oContact.UserProperties.Add("InsuranceCompany", olText)
```

When you add a custom property to a Contact object, by default, it's added to all Contact objects. You can add it to just the specified Contact by passing .F. for the Add method's optional third parameter.

Once you've added the property, you can set it by assigning a value to its Value property:

```
oContact.UserProperties["InsuranceCompany"].Value ;
   = "MegaHuge"
```

Similarly, to read the value that's there, access the Value property:

```
REPLACE cInsurance WITH ;
   oContact.UserProperties["InsuranceCompany"].Value
```

Interestingly, the Find method works on the UserProperties collection. You can use it to determine whether a particular property exists:

```
oProperty = ;
   oContact.UserProperties.Find("InsuranceCompany")
```

The method returns an object reference to the user property (which is, in fact, an object). Then you can work with it directly rather than through the object hierarchy:

```
IF VarType( oProperty ) <> "X"
   oProperty.Value = "Local Insurance Company"
ENDIF
```

As with other Outlook automation, you need to call the Save method for the Contact item in order to have the new property and any value you've given it preserved:

```
oContact.Save()
```

Finally, while I've been talking only about Contact items here, almost everything in this answer works equally well with the other Outlook items. They all have UserProperties collections, they support the Find method, and so forth. Once you master these techniques for Contacts, you can use them for mail messages, task list entries, calendar entries, and the rest.

–Tamar