COM302:

Automating Outlook with Visual FoxPro

Ted Roche, MCSD, MCSE, VFP MVP
Ted Roche & Associates, LLC
http://www.tedroche.com

Microsoft Outlook is a personal information manager with the ability to store contacts and notes; journal activities; track appointments; provide workflow routing; use custom forms; interact with other applications using VBA, COM, VBScript, MAPI, and other extensions; display HTML; become part of a digital dashboard; present functionality on Web pages via Web Parts ActiveX controls; and read and write e-mail from Internet-standard or Exchange mail servers. Visual FoxPro can interact with Outlook in a number of ways, either by driving the process or being called via Automation. In addition, VFP 7 lets you hook into the Outlook event model and fire VFP code in response to Outlook events. This session reviews the object models, highlights differences between the many versions of Outlook, and demonstrates VFP-Outlook integration using each of the interfaces.

Introduction

Outlook is a powerful application. Besides a very interactive interface, it has the ability to work with different backend mail servers, store its data either on that server (Exchange) or in its own local cache. It can fire reminders, track changes to Office documents (journaling), track names, addresses, places, link to interactive maps and compose email and word processing documents. For many offices, Outlook is all they need to manage their business. For many other businesses, Outlook supports the miscellaneous business functions, while dedicated applications handle other aspects of the business, such as accounting, sales, inventory control or manufacturing. In these latter situations, there will often be interest in combining or integrating some of the Outlook functions with other applications.

As a FoxPro developer, either an employee or consultant to the business, you are in a unique position to evaluate the needs of the business and make a professional recommendation on the way to proceed. In this document, we focus on the mechanics of working with Outlook. However, just because you can do something doesn't mean that you should. Carefully consider the needs of the business, both current and future, and consider the alternatives that may be available. If the company needs a Customer Relationship Management (CRM) system, you can cobble one together from scratch, but often an off the shelf solution is available cheaper and sooner. The unique power of Visual FoxPro is to fit into niches where other tools are too weak or too strong, too limited or too comprehensive.

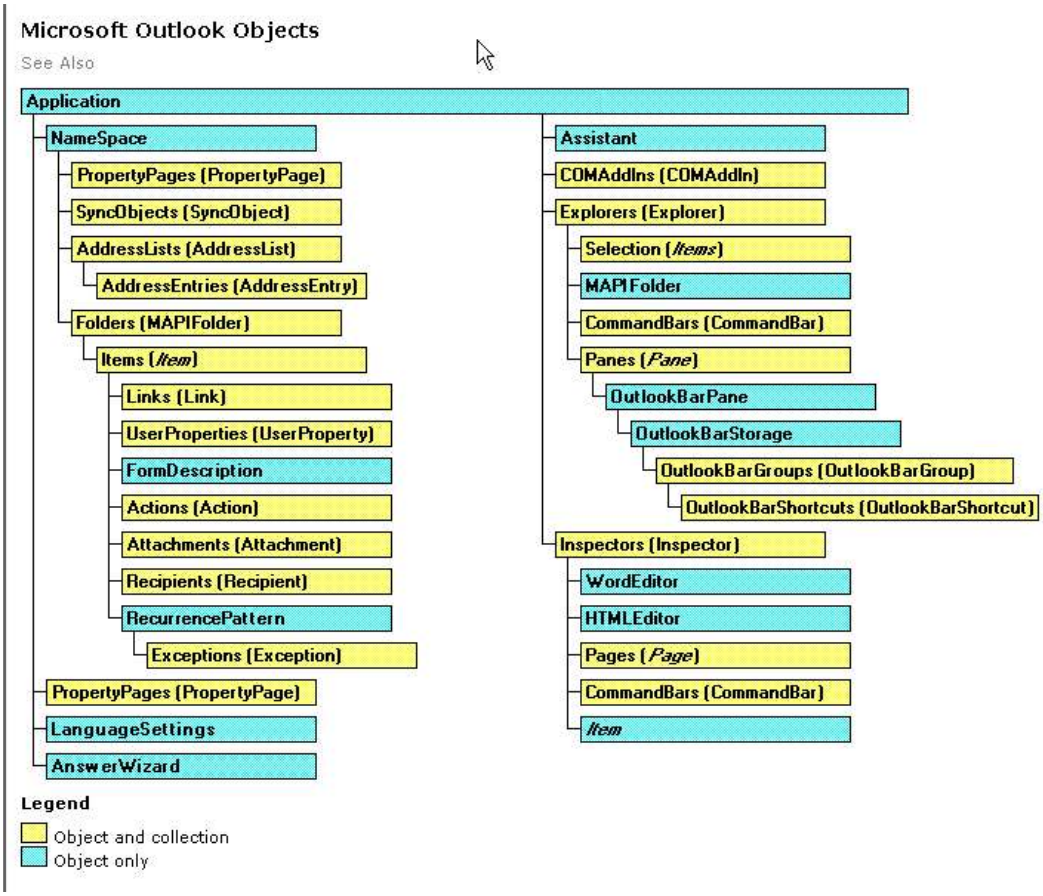

Figure 1: The Outlook Object Model: The Application contains the Namespace, the NameSpace contains the Folders, the Folders contain the Items, the Items contain the properties, attachments and recipients. (from the Microsoft Outlook 2000 Help File)

Exploring the Outlook Automation Model

Documentation on working with Outlook using VBA and VBScript is available directly from within the Outlook Help file. Depending on your version of the product, you'll find the help under topics such as "Advanced Customization." The Outlook Object Model (see Figure 1) is helpful to understand the relationship between the components of Outlook.

Basic Automation

The simplest of Automation can happen in less than a dozen lines of code, the standard "how to send an email." See Listing 1 for the sample code.

```
*=============================================================================
* Program:               AUTOLOOK.PRG
* Purpose:               Automate Outlook
* Author:                      Ted Roche
* Copyright:                   (c) 2002
* Last revision:         07/22/02
*=============================================================================

#INCLUDE "MSOutl9.H"

LOCAL loOutlook as Outlook.Application, ;
      loNameSpace as Outlook.NameSpace, ;
      loFolder as Outlook.MAPIFolder, ;
      loMailItem as Outlook.MailItem

loOutlook = CREATEOBJECT("Outlook.Application")
loNamespace = loOutlook.GetNamespace("MAPI")
loFolder= loNamespace.GetDefaultFolder(olFolderInbox)
loMailItem = loFolder.Items.Add(olMailItem)
WITH loMailItem
  .To = "ProFox@Leafe.Com"
  .Attachments.Add(HOME()+"Fox.bmp")
  .Display()
ENDWITH
```

Listing 1: Sending a simple email can be done in a dozen lines of code

Reviewing the code in Listing 1 can cover a lot of the basics of Automation.

The INCLUDE pre-processor command allows the programmer to substitute easily readable pre-defined constants for obscure numeric values. The header file can be created automatically in VFP 7 or later using the Object Browser, or with earlier versions using a third-party tool such as Rick Strahl's GetConstants.exe utility (http://www.west-wind.com/Webtools.asp).

The LOCAL command sets the scope of all the variables, avoiding contention with variables of the same name declared elsewhere, and also declares their type for use by IntelliSense. IntelliSense is enormously helpful in speeding typing, avoiding spelling errors, and prompting you for correct method and property names.

The next four lines create a hierarchy of objects: first, an instance of the Outlook application, then declaring the proper namespace (so that MAPI commands are properly recognized), then pointing to a default folder, and finally, adding a new item (or type olMailItem) to that folder. We have now created a new email item.

Finally, properties are set and the email can be displayed by calling the Display() method or sent with the Send() method.

Which Outlook is which?

Writing generic Outlook code is challenging when there are so many versions of Outlook available.

- Outlook Express is not Outlook – it is an entirely different program, originally named Internet Mail and News, built by a different team within Microsoft.
- Outlook 97 and 2000 came in two varieties – the Internet Mail or Corporate Workgroup ("C/W") varieties. Despite their names, either works with Internet Mail. However, MAPI and Extended MAPI interfaces only work with the C/W version.
- In Outlook XP, the Internet and Corporate versions are combined, with the loss of some nice features from each, but with the gain of a single set of objects to code against.

When you and your employer or client decide to create an application that uses Outlook, you'll need to ensure that you can identify the targeted versions, and make appropriate plans to support different versions or upgrades.

Microsoft's Security Patches

In response to the ILOVEYOU email worm and numerous other scripting exploits, Microsoft issued a series of Security Update Patches that restricted the types of attachment extensions that Outlook will allow, and restricted access to the Outlook address book and the ability to send email using Automation. The security patches were integrated into Outlook 2000 in Service Pack 2 and are built into Outlook XP.

Because of the ease of sending mail using the Outlook Object model, evil script kiddies developed a series of harassing scripts to ransack a users Outlook Address Book and spawn email worms that cost businesses thousands of dollars to clean up. (Maybe "billions" – seehttp://news.com.com/2100-1001-240112.html?legacy=cnet) . Worms like Melissa and ILOVEYOU are a pain for all of us, so Microsoft solved the problem with a series of security patches for older systems and by integrating in new security measures for the newer systems. Here are a few scenarios when the security patches are likely to trip you up.

When attempting to send mail on a patched system, you'll see a dialog like Figure 2. There is a five second delay before the "Yes" button is enabled on the dialog.

Figure 2: There is a 5-second delay before the "Yes" button enables, thanks to this security patch.

Whenever you attempt to access the Address Book (with code like (**`MailItem.Recipients.Item(1).Resolve()`**), a dialog box (see Figure 3) will appear to let the user know that an application is manipulating their email addresses. This time, instead of a 5-second pause, the dialog allows a single access (by not selecting the checkbox) or a timed access of 1, 2, 5 or 10 minutes between dialogs.

Figure 3: The clock's running for how long you access the Address Book

Working with the Outlook Security Patches

Developers will have to choose the option that makes the most sense for them in working around the security patches.

- A dangerous course would be to consider locking down development at the Office 2000 SR-1 version, and not installing any further patches. That leaves the installation open to future exploits, and is almost sure to fail when someone runs Windows Update or purchases a new machine.
- Nearly as dangerous would be to ship your application as is, instructing the clients to click the warning dialogs when they appear. Getting the users in the habit of clicking virus-warning dialogs is a ***really*** bad idea, and will surely lead to spreading of worms and viruses.
- Installations that use both Outlook and Exchange have some options at the Exchange administrator level to tone down the warnings, or register certain applications as safe. Details are in the Microsoft Knowledgebase. See the links at the end of this paper.
- Outlook Redemption is a third-party COM object that can easily be added to your Automation code. It simulates much of the existing Automation interface, but uses Extended MAPI to manipulate the underlying objects, rather than using the Automation interface. Because scripting languages cannot access Extended MAPI, nor can they use the Redemption COM object, there is no danger of security compromise with this solution. In addition, the Redemption MAPIUtils provides some features missing from the Automation interface. This is a commercial product, with a reasonable price for it's remarkable features. See the References section below for more information.
- The Express ClickYes program is a different technique to solve the problem. It is a small resident program that can be turned on and off via some Windows API calls. When enabled, it will spot the dialog and click the "Yes" button, even before the dialog becomes visible. The price is right – it's free. See the contact information in the Reference section below.

Outlook Collection and FOR EACH Issues

Nearly every group of objects you try to access in Outlook is represented by a collection. A collection can be thought of as an array of pointers to individual objects, but that metaphor is not exactly true, and where it breaks down can actually cause a FoxPro programmer grief. An OLE collection is actually a *property* of an object that fires an *access method* when it is called. That access method returns a collection of objects. Since the access method is fired each time the property is accessed, a potentially different collection can be returned each time. The solution to this is to access the collection only once, and preserve the object pointer to that collection to use for future use.

```
* The Right Way to access a collection:
```

```
loItems = ;
  loOutlook.NameSpace("MAPI").GetDefaultFolder(olFolderInbox).Items
FOR m.i = 1 to loItems.Count
  loMailItem = loItems(m.i)
   * Process item…
NEXT

* The Wrong Way to access collection:
FOR m.i = 1 to ;
  loOutlook.NameSpace("MAPI"). ;
  GetDefaultFolder(olFolderInbox).Items.Count
  loMailItem = ;
    loOutlook.NameSpace("MAPI"). ;
      GetDefaultFolder(olFolderInbox).Items(m.i)
   * Process item…
NEXT
```

Listing 2: Correct and incorrect ways to manipulate a collection in Outlook

In Visual FoxPro 5, 6 and 7, there have been issues with using the FOR EACH clause when iterating through a COM collection object. In some circumstances, an object reference is not being properly incremented or decremented, and instability (crashes) result. My advice is to replace code such as this:

```
FOR EACH loMailItem in loMailItems
   …* Do your processing here…
NEXT
```

with:

```
FOR m.i = 1 to loMailItems.Count
  loMailItem = loMailItems(m.i)
   … * Do your processing here…
NEXT
```

Four Models of Automation

There are three modes of interacting with Outlook: VFP can drive Automation of Outlook, Outlook can drive the Automation, invoking a VFP component, or the two applications can be bound via Event Binding. In the next sections, we'll look at each of these and evaluate the situations in which each is appropriate.

VFP Driving

The nice thing about VFP driving the operation is that we, as FoxPro developers, are in charge. Alan Cooper described this technique in his book *The Inmates Are Running The Asylum*. The problem with this technique is similar to the paradigm shift of moving from a procedural coding model to an event-driven one. In most cases, when developing an interactive application involving Outlook, *we're not in charge; the users are.*

The advantages of using VFP to drive the operation are that we are working in a familiar, rich and powerful development environment, are able to exercise fine control programmatically and in error-handling, and have the blazing speed of Visual FoxPro. For tasks that involve behind-the-scenes programming, or launching an Outlook form and then letting it go, driving from VFP is the right solution.

See the example above in "Basic Automation" for a good example of VFP driving the process.

Outlook Driving

When Outlook is driving the operation, the client is typically working in the full Outlook environment, and launches a form, clicks a button, or invokes a macro that causes VFP code or data to be invoked. Outlook is

a somewhat constrictive development environment, if you are coming from a full-fledged development tool, but there is a lot that can be done with it.

Outlook has a built-in Forms Designer. Forms are one-off items, with no base classes or inheritance. Outlook forms can contain some basic controls (buttons, textboxes, editboxes) Outlook forms can have VBScript code associated with them. There is a Scripting debugger available from http://www.microsoft.scripting, the home page for a lot of resources and downloads on all things VBScript.

Outlook does not have a macro recorder, like many of the other Office programs. There is, however, a VBA Editor and development environment. Outlook has the curious design of allowing only one VBA project to be created and associated with a given instance of Outlook, so if your client already has such a project, you will need to try to merge the code snippets to get two projects to run at once.

There are many ways for Outlook to drive the process of Automation, and there are a number of fine books on the subject of creating Outlook applications with Outlook as the focus and center of the development model. I strongly encourage you to pick up one or more of these, as they speed the process of development immensely.

The example I'll provide here is using a custom Outlook form with attached VBScript code to invoke a VFP DLL and record what was sent in the message.

1. Create a new Outlook Form with Tools | Forms | Create a Form
2. Add your custom controls and fields to it. In the example, I create a Message ID field to track the message
3. Add VBScript to the Form by selecting Form | View code from the Form Designer.
4. Save the form to disk (it has an OFT extension, for Outlook Form Template?) or to the Personal or Organizational Form Library. The form must be in a library to avoid the "This form has macros – enable?" safety message on startup.
5. Create the form out of your VFP application by launching it with **`loOutlook.CreateItemFromTemplate(<full path to template>)`**, if creating it from a file-based object, or **`loFolder.Item.Add()`** passing it the Outlook form name, typically something like "IPM.Note.YourForm" – the name you assign it when you save it to a library.

Typical code that you'll create will be able to manipulate the object that you are working with (Item is the equivalent of THIS) and the environment, through Item.Application. In the example in Listing 3, I invoke a FoxPro DLL, concatenate some strings and a custom item on the form, and send the entire thing to the DLL. The DLL is a simple demonstration of technology that just logs the message to disk. It does bring up one good point though, in that the log file can be a little hard to find. On my machine, it's located in C:\Program Files\Common Files\System\Mapi\1033\NT, so make sure you set the default directory of your DLL (perhaps by reading a setting from the registry) before you make any assumptions about your path.

```
Option Explicit
Function Item_Send()
  ' ************************************************************
  ' VBScript Item_Send
  ' Written by: Ted Roche, MCSD, MCSE, http://www.tedroche.com
  ' Copyright 2002 by Ted Roche
  ' Purpose: Invoke a VFP DLL to journal sent mail and attachments
  ' Date: February 2, 2002 (02/02/2002)
  ' Version: 1.2
  ' Modifications:
  ' ************************************************************

  dim VFPDLL, cMessage, lReturn, nRetVal, cMessageID
  cMessage = ""

  ' ************************************************************
```

```
  ' Create the message string by adding fields
  ' ************************************************************

  cMessage = cMessage & "To: " & Item.To & CHR(13)
  cMessage = cMessage & "CC: " & Item.CC & CHR(13)
  cMessage = cMessage & "BCC: " & Item.BCC & CHR(13)
  cMessage = cMessage & "Subj: " & Item.Subject & CHR(13)
  cMessage = cMessage & "Sent: " & Now & CHR(13)
  cMessage = cMessage & "Body: " & Item.Body & CHR(13)

  ' ************************************************************
  ' Get the Message ID from the custom form
  ' ************************************************************
  ' The MessageID is a custom field on the form. It's set by FoxPro
  ' code when the form is invoked.

  cMessageID = Item.UserProperties("MessageID").Value

  ' ************************************************************
  ' Invoke the VFP Object
  ' ************************************************************

  ON ERROR Resume Next
  set VFPDLL = CreateObject("VFPDemo.LogMail")

  ' ************************************************************
  ' Journal the message
  ' ************************************************************

  IF Err.Number = 0  THEN ' No error
    lReturn = VFPDLL.mUpdate(cMessageID, Item.Subject, cMessage)
  END IF ' Err.Number = 0

  SET VFPDLL = Nothing

  ' ************************************************************
  ' Display an error if appropriate
  ' ************************************************************

  IF Err.Number > 0 THEN
    DIM cErrMessage, cErrTitle
    cErrMessage = "An error occurred journalling this entry. "
    cErrMessage = cErrMessage & "The message will be sent, but no
journal entry was made. "
    cErrMessage = cErrMessage & "Contact technical support to report
this problem. "
    cErrMessage = cErrMessage & "Error # " & CStr(Err.Number) & ": "
    cErrMessage = cErrMessage & Err.Description & " " & Err.Source
    cErrTitle = "Journalling error"
    nRetVal = MsgBox(cErrMessage , vbOKOnly + vbCritical +
vbDefaultButton1, cErrTitle)
  END IF ' Err.Number > 0

End Function
```

Listing 3: VBScript code to invoke a VFP DLL upon sending a message

Drag and Drop

Supporting drag and drop from Outlook into VFP can be surprisingly easy. OLE Drag and Drop from Outlook only supports a text object, and that object can't give you enough information to actually uniquely identify the item (or items) that are dragged from Outlook. The text will depend on the order of columns in the user's Outlook view, but is usually tab-separated text of "From","Subject","Date" and separate lines of tab-separated text for each message.

The trick is to ignore the dragged object altogether, and query Outlook directly. An entire example is included on the conference CD (DropMail.*), but the key is to use the ActiveExplorer property. Explorers and Inspectors in Outlook are the equivalent of Explorer interfaces and forms in VFP. You can access these to determine what information is displayed to the user and react appropriately. In the case of drag-and-dropping, the files most recently selected by the user to initiate the action are selected in the currently active Explorer. All that is needed to retrieve a handle to those objects is the following code snippet:

```
loOutlook = CREATEOBJECT("Outlook.Application")
loSelection = loOutlook.ActiveExplorer.Selection
```

Event Binding

Event Binding is a new feature of Visual FoxPro 7.0. Event binding allows developers to register their programs as recipients of event messages for other applications, binding an instance of their event handler to specific events in another application, and causing code to run when those events occur.

See the EventBound code for a simple demonstration of events in Outlook directly firing code in Visual FoxPro.

And clean up your room!

Email data is not pretty data. You've all received emails where a message was embedded into another and another… or the one that includes all the CCs of the message and a one-line joke. Email is like that. If you are sorting out email addresses or processing through directories, you have to anticipate that the data you'll run into will be dirty, at least, and quite likely not what you expect. Email addresses will have every mixture of cases, people will regularly change their address, and you will find object in email folders that will surprise you. Here are a couple of things to look out for:

1. Messages added to a folder while you're processing
2. Incorrect data types for objects in the collection
3. Bogus email addresses

A routine for checking email addresses to ensure they follow some basic rules is included in the conference notes, as EmailCheck.PRG, courtesy of Wayne Willingham.

LookOut!

As you've seen in this brief tour of Outlook, there are a lot of ways that Outlook and Visual FoxPro can interact. The key to finding the optimal way to work with Outlook is to examine the problem your employer or client needs to solve, and working through the architecture that's right for them. Use a VFP-Driven approach when you want to process through material in the background, or when you just want to launch a form and let it go. Use the Outlook-driven or event-bound technique to let the operators initiate the program code based on their actions. Use a drag-and-drop technique for more graphical and interactive portions of the application. Think carefully about security, and evaluate the appropriateness of your solution based on the risks to your client.

About the Author

Ted Roche develops Web, client-server and LAN-based applications using Microsoft Visual FoxPro, Microsoft SourceSafe, SQL Server, and other best-of-breed tools. He is a principal in Ted Roche & Associates, LLC, http://www.tedroche.com, based in New Hampshire, USA where he offers consulting, training and mentoring as well as software development services. Ted is author of Essential SourceSafe, co-author of the award-winning Hacker's Guide to Visual FoxPro 6.0, and a contributor to five other FoxPro books. In addition to numerous magazine articles, he is a popular speaker at conferences worldwide. Ted is a Microsoft Certified Solution Developer, Microsoft Certified System Engineer, and eight-time winner of the Microsoft Support Most Valuable Professional award.

Books

Byrne, Randy *Building Applications with Microsoft Outlook Version 2002*, Microsoft Press, 2001; ISBN: 0735612730

Rizzo, Thomas, *Programming Microsoft Outlook and Microsoft Exchange*, Microsoft Press, 2000; ISBN: 0735610193

Slovak, Ken, Chris Burnham, Dwayne Gifford, *Professional Outlook 2000 Programming : With VBA, Office and CDO,* Wrox Press, 1999, ISBN: 1861003315

Other References and Resources

Outlook Redemption can be found at http://www.dimastr.com/redemption/. Written by Outlook MVP Dmitry Streblechenko, it slips between your code and Outlook, emulating all of the "regular" COM calls directly through to the Outlook interface, but using Extended MAPI to handle those calls which would normally generate a security warning from Outlook's security patches. Great stuff, and stable.

Wayne Willingham wrote EmailCheck.PRG, supplied with the conference materials. Wayne is the Director of Information and Technology for Digital Alchemy, Inc., a leader in hospitality-specific Customer Relationship Management at http://www.data2gold.com/ and an activist in the Dallas / Fort Worth FoxPro User Group at http://www.dfwfox.org/

Information on customizing the security restrictions in Outlook (for Exchange server customers only) is available at http://www.microsoft.com/Office/ORK/2000/journ/outsecupdate.htm . More information, with links to another half-dozen articles, can be found at: http://support.microsoft.com/support/kb/articles/Q262/6/31.asp

The premier place on the web for great information on programming Outlook and Exchange is http://www.slipstick.com. Great information on Outlook and Exchange, with extensive links to everything else on the subject, can be found at Sue Mosher's site. Her free email newsletter, Exchange Messaging Outlook (EMO) has a special article on Outlook Redemption at http://www.slipstick.com/emo/2001/up010228.htm.

Woody Leonhard publishes a series of email newsletters on Windows, Office, Project and other topics. In this issue, he talks about the first of the Outlook security patches. While his opinions are sometimes extreme, there is a lot of good information in his columns. Check out this classic at: http://www.woodyswatch.com/office/archtemplate.asp?v5-n27

"A New Outlook," Andrew Ross MacNeill, *FoxPro Advisor*, June 2000

"Out-Foxing Outlook," Andrew Ross MacNeill, *FoxPro Advisor*, September 2000

"Exposing Outlook with Visual FoxPro," Andrew Ross MacNeill, *FoxPro Advisor*, October 2000

"Outlook Update," Andrew Ross MacNeill, *FoxPro Advisor*, June 2001

"Automate Outlook with Visual FoxPro," Art Bergquist, *FoxPro Advisor*, August 2001

Requests for Comments (RFCs) are the Internet's working standards. Visit the authoritative source at http://www.ietf.org/rfc.html. Make sure you check the index to ensure that the standard you find hasn't been updated or deleted.

The MailTo: URI is an example of an RFC. http://www.ietf.org/rfc/rfc2368.txt specifies that it can be any set of key-value pairs, but "The creator of a mailto URL cannot expect the resolver of a URL to understand more than the "subject" and "body" headers."

The Exchange SDK and Platform SDK are essential for mastering the One Microsoft Way of doing email. Look for them at http://www.microsoft.com/exchange and http://msdn.microsoft.com, respectively.

Express ClickYes is a program you can control to bypass the Outlook Security dialogs. The price is right – free. http://www.express-soft.com/mailmate/clickyes.html. An example in VFP is on the web site.

Outlook Redemption is powerful set of tools to once again regain control of Outlook object automation after applying the security patches: http://www.dimastr.com/redemption for details. This goes far beyond the ClickYes program in providing functionality for many items that are beyond the reach of Automation by using Extended MAPI to manipulate the email.