Email and File Transfer the Fast (and Cheap!) Way

Doug Hennig

In the previous two issues, Doug discussed free libraries generously provided by Craig Boyd to compress and decompress files and encrypt and decrypt strings and files. In this issue, he discussed two more libraries that add MAPI email and file upload and download capabilities to your applications.

Emailing from VFP

Emailing from an application is a very common thing to do these days. Maybe your users want to automatically send invoices to their customers. Perhaps the application should send you an email when an error occurs. Regardless of the reasons, how to send emails from a VFP application is a commonly asked question in VFP forums.

Craig Boyd thoroughly explored various means of emailing from VFP in a nine-part series of blog posts; search his blog (http://www.sweetpotatosoftware.com/spsblog) for "email" to access these articles. In his articles, he discusses open source solutions but there are numerous commercial products available as well.

One problem you'll encounter if you use MAPI (Mail Application Programming Interface) to email and you have Microsoft Outlook installed is the dreaded Outlook security dialog shown in **Figure 1**. The purpose of this dialog is to prevent rogue applications from sending emails containing malware from your account. Of course, it also causes problems for legitimate applications as well. Although there are a couple of solutions for this issue, including Click Yes and Redemption, in my opinion they're kludges.

Once again, Craig Boyd comes to the rescue. He created a library called VFPExMAPI.FLL that provides an interface to Extended MAPI, a mechanism for using MAPI clients like Outlook to work with email. Craig's library only concerns itself with sending messages, not retrieving or reading them. However, one of the cool things about his library, besides the fact that it's free, easy to use, and has a small footprint (only 36K), is that it avoids the Outlook security issue.

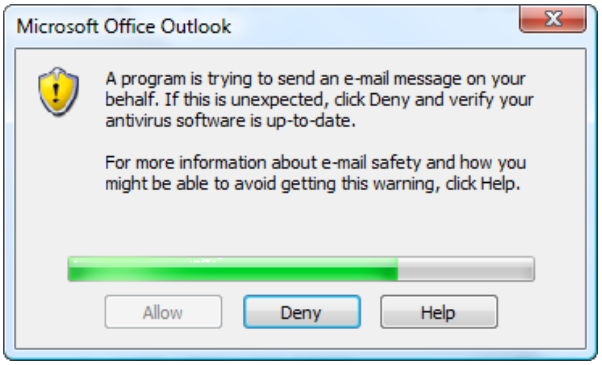

Figure 1. The dreaded Microsoft Outlook security dialog.

Here's a list of blog entries Craig has posted about VFPExMAPI.FLL. You can read them all if you wish or jump to the first one for the most current download and documentation.

- http://tinyurl.com/2wqdudm
- http://tinyurl.com/34oz9np
- http://tinyurl.com/38gou7z

Using VFPExMAPI.FLL

Like other libraries, the first thing to do is open it: SET LIBRARY TO VFPExMAPI.FLL.

To create a plain text message, use EMCreateMessage(cSubject, cBody, nImportance). For nImportance, use 0 for Low Importance, 1 for Normal Importance, and 2 for High Importance. If you'd rather use HTML or RTF for the body, use EMCreateMessageEx instead. Both of these functions return .T. if they succeeded.

After creating the message, specify recipients using EMAddRecipient(cEmailAddress, nType). For nType, use 1 for To, 2 for CC, and 3 for BCC. Another value, 0 for original, is supported but I couldn't figure out what that's for. Call EMAddRecipient once for each recipient. This function returns .T. if it succeeded.

Use EMAddAttachment(cFile) to add an attachment. cFile doesn't have to be a full path; it just has to be in the current directory or VFP path. EMAddAttachment returns .T. if it succeeded.

Finally, to send the message, call EMSend(lSendImmediately). If you omit the parameter or pass .F., the message sits in the Outbox of their email program until the next time a send and receive operation executes. Pass .T. to send the message immediately. Note that this doesn't work for Microsoft Outlook 2002 and 2003; it acts like you passed .F. See http://tinyurl.com/26j2fyp for details. EMSend returns .T. if it succeeded.

Here's an example that sends two plain text messages: one with an attachment and one without (please use your email address instead of mine!):

```
#define MAPI_TO 1
#define IMPORTANCE_NORMAL 1

* Send a plain text message.

lcRecipient = 'dhennig@stonefield.com'
lcBody = "Here's a plain text email."
EMCreateMessage('Test message #1', lcBody, ;
    IMPORTANCE_NORMAL)
EMAddRecipient(lcRecipient, MAPI_TO)
llResult = EMSend()
    && don't send yet, just put it in the
    && Outbox
messagebox('Check the Outbox then click OK')

* Send a plain text message with an
* attachment.

lcBody = "Here's a plain text email " + ;
    "with an attachment."
EMCreateMessage('Test message #2', lcBody, ;
    IMPORTANCE_NORMAL)
EMAddRecipient(lcRecipient, MAPI_TO)
lcFile = 'Attachment1.txt'
strtofile('This is my attachment', lcFile)
EMAddAttachment(lcFile)
llResult = EMSend(.T.)
    && send both messages now
messagebox('Check the Outbox and the Inbox')

* Clean up.

erase (lcFile)
```

This example sends a message formatted as HTML:

```
#define MAPI_TO 1

#define IMPORTANCE_HIGH 2

* Send an HTML message.

lcRecipient = 'dhennig@stonefield.com'
text to lcBody noshow
<html>
<body>
<p>Here's an HTML email.</p>
<ul>
  <li><a href="http://www.swfox.net">Southwest
Fox</a> is great!</li>
  <li>Check Craig Boyd's <a href=
"http://www.sweetpotatosoftware.com/SPSBlog/">
blog</a></li>
</ul>
<p>Sent from a VFP application</p>
</body>
</html>
endtext
EMCreateMessageEx('Test message #3', lcBody, ;
    IMPORTANCE_HIGH)
EMAddRecipient(lcRecipient, MAPI_TO)
llResult = EMSend()
    && don't send yet, just put it in the
    && Outbox
messagebox('Check the Outbox')
```

VFPExMAPI.FLL has a few other functions. EMDisplay displays the user's email editing dialog (the same dialog that appears when you create a message) and returns .T. if they sent the message or .F. if they closed the dialog without sending. EMBindEvent and EMUnbindEvent are supposed to allow you to bind to email events, such as messages arriving, but I got a "Function argument value, type, or count is invalid" error every time I tried to use it. Passing a string as a second parameter to EMBindEvent prevented that error but then caused VFP to crash when Outklook received an email. Obviously, Craig's documentation is missing some information about this function so until he updates the documentation, you're best avoiding it.

File transfer

Modern applications often need to transfer files to and from web or FTP sites. For example, one strategy for keeping an application up-to-date is:

- The application downloads a file from a web site containing information about the latest version, such as a version number.
- If the version of the running application is earlier than the latest version, download an installer for the new version, launch the installer, and terminate the application.

Although there are numerous commercial libraries that provide file upload and download features, Craig Boyd's VFPConnection library is simple and even better, free. Here's a list of blog entries Craig has posted about VFPConnection.FLL. You can read them all if you wish or jump to the last one for the most current download and documentation.

- http://tinyurl.com/3aamldr
- http://tinyurl.com/33w4dul
- http://tinyurl.com/333d4b5

Using VFPConnection.FLL

Before using VFPConnection functions, open the library using SET LIBRARY TO VFPConnection.FLL.

To download a file from an FTP site, use FTPGet(cSourceURL, cDestinationFile [, cProgressCallback [, cTraceCallback]]). If you

need to specify a user name and password or port, include them in cSourceURL using the format username:password@ftpsite:port/filepath; for example, "FTP://MyUserName:MyPassword @MySite.com:21/MyFolder/SomeFile.txt." I'll discuss the two callback parameters later.

According to Craig's documentation, FTPGet returns .T. if it succeeds or .F. if it fails. However, my testing showed that it may return .T. even if it fails. If the failure is due to a connection problem, it returns .F. However, if some other problem occurs, FTPGet creates the file anyway and returns .T. The file contains HTML indicating the problem. For example, attempting to retrieve a file that doesn't exist on the server results in a file with this content (formatted here for readability):

```
<html>
<head>
<title>Error</title>
</head>
<body>
The system cannot find the file specified.
</body>
</html>
```

This complicates downloads a little: even if FTPGet returns .T., you need to check the content of the file to see if the download succeeded or not.

To download a file from a web site, use HTTPGet; it accepts the same parameters and returns the same results.

Here's an example that downloads a ZIP file and a web page from my web site:

```
lcWebSite = 'http://www.stonefield.com/'
lcFileToGet = 'TaskScheduler.zip'
lcPageToGet = 'techpap.aspx'

* Download a file using FTP.

llResult = FTPGet(lcWebSite + 'pub/' + ;
    lcFileToGet, lcFileToGet)
messagebox('Downloading ' + lcFileToGet + ;
    ' from ' + lcWebSite + ;
    iif(llResult, ' succeeded', ' failed'))

* Download a file using HTTP.

llResult = HTTPGet(lcWebSite + lcPageToGet, ;
    lcPageToGet)
messagebox('Downloading ' + lcPageToGet + ;
    ' from ' + lcWebSite + ;
    iif(llResult, ' succeeded', ' failed'))
```

Rather than writing the downloaded file to a file on disk, you can instead store it to a string by calling FTPToStr(cSourceURL [, cProgressCallback [, cTraceCallback]]) or HTTPToStr (same parameters). Both return the file as a string instead. They are the equivalent of calling FTPGet or HTTPGet followed by FILETOSTR() to read the downloaded file into a string, but perform better since there's no disk read or write involved.

Here's an example similar to the earlier one but using FTPToStr and HTTPToStr instead:

```
lcWebSite = 'http://www.stonefield.com/'
lcFileToGet = 'TaskScheduler.zip'
lcPageToGet = 'techpap.aspx'

* Retrieve a file directly to a string.

lcFile = FTPToStr(lcWebSite + 'pub/' + ;
    lcFileToGet)
messagebox(transform(len(lcFile)) + ;
    ' bytes were retrieved from ' + ;
    lcFileToGet)

* Retrieve a page directly to a string.

lcPage = HTTPToStr(lcWebSite + lcPageToGet)
messagebox(transform(len(lcPage)) + ;
    ' bytes were retrieved from ' + ;
    lcPageToGet)
```

There are versions of these functions that use SSL for the connection: FTPSGet, HTTPSGet, FTPSToStr, and HTTPSToStr. There's also a version for retrieving a file from a local or remote drive: FileGet. Specify the source file as "FILE://filepath."

Callback during download

Downloading a large file or even a small one over a slow connection can take a while, during which the user may think your application has hung. Providing download feedback such as a progress dialog assures the user something is happening.

All of the functions discussed so far accept a couple of optional parameters: functions or methods to call when bytes have been retrieved (the cProgressCallback parameter) and when information is available about the communication between the client and the server (the cTraceCallback parameter). When it calls the progress callback function, VFPConnection creates two public variables, nConnectTotalBytes and nConnectBytesSoFar, your function can use to display or log progress. When it calls the trace callback function, which is does at events such as when attempting to make the connection and when the connection has been made, it creates two variables, nTraceDataType and cTraceData, your function can use to display or log progress of events other than the actual download. nTraceDataType indicates the type of information available in cTraceData: 0 = text, 1 = header in, 2 = header out, 3 = data in, 4 = data out, 5 = SSL data in, 6 = SSL data out, and 7 = end.

The following example is similar to code we saw earlier but uses a progress meter in a dialog (see **Figure 2**) to show the download progress:

```
loProgress = newobject('ProgressForm', ;
    'Samples.vcx')
loProgress.Show()
FTPToStr(lcWebSite + 'pub/' + lcFileToGet, ;
```

```
'loProgress.Update()')
```

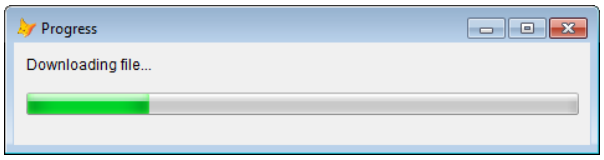

Figure 2. You can use a callback function to display the progress of a file download.

The Update method of the form updates the Ctl32_ProgressBar control on the form using this code:

```
This.oProgress.ctlValue   = nConnectBytesSoFar
This.oProgress.ctlMaximum = nConnectTotalBytes
```

Uploading files

VFPConnection has functions that mirror the download functions but upload instead: FTPPut, HTTPPut, FTPSPut, HTTPSPut, and FilePut. These functions essentially reverse the first two parameters: the first is the name and path for the local file and the second is the URL for the destination on the server, including username, password, and port if necessary. Like their "Get" counterparts, these functions accept progress and trace callback functions.

Other VFPConnection functions

If you need to post data to a web site, VFPConnection has a couple of functions for you: HTTPPost(cPostURL, aPostData [, cProgressCallback [, cTraceCallback]]) and HTTPSimplePost(cPostURL, cPostData [, cProgressCallback [, cTraceCallback]]). For the aPostData parameter, pass HTTPPost a two-dimensional array by reference (that is, prefixed with @) containing the name of the post parameters in the first column and the values in the second. For HTTPSimplePost, specify the post parameters using the same syntax you'd use in a URL (Parameter1Name=Value1&Parameter2 Name=Value2 …) for the cPostData parameter.

The FTPCommands and FTPSCommands send a series of FTP commands to an FTP site, providing you with the same type of commands an FTP client application typically has, such as creating or renaming directories. These functions accept a URL, an array of FTP commands (passed by reference), and progress and trace callback functions.

URLEncode and URLDecode accept a string and return URL encoded or decoded versions of that string. Encoding a string means converting any character other than letters and numbers to their encoded equivalents, which is "%" followed by the character's ASCII value in hex (such as "%20" for a space). Decoding converts encoded characters in a string back to their normal equivalents. Here's an example that uses URLEncode to download file with a space in the name:

```
lcWebSite = 'http://www.swfox.net/'
lcFileToGet = 'Vendor Prospectus.pdf'

* Download a file with a space in the name.

lcResult = FTPToStr(lcWebSite + lcFileToGet)
messagebox(lcResult, 0, 'Download Failed')

* Use URLEncode to resolve the problem.

llResult = FTPGet(lcWebSite + ;
    URLEncode(lcFileToGet), lcFileToGet)
```

SetConnectTimeout(nSeconds) allows you to adjust how long VFPConnection waits to connect to a URL; the default is 10 seconds. Similarly, SetResponseTimeout(nSeconds) affects the timeout for all other VFPConnection operations.

DateStrToEpochSec(cDateTime) converts a date time value as a string to the number of seconds since midnight of January 1, 1970. This is useful because there seems to be as many ways to express a date time value as there are web sites. For example, RSS files use RFC 822 format, such as Sun, 28 May 2006 20:48:40 GMT. The following code downloads a page from VFP developer Joel Leach's blog and converts the latest publication date to a VFP DateTime:

```
lcPage = HTTPToStr('http://weblogs.foxite' + ;
    '.com/joel_leach/rss.aspx')
lcPub = strextract(lcPage, '<pubDate>', ;
    '</pubDate>')
ltPub = DateStrtoEpochSec(lcPub) + ;
    {^1970-01-01 12:00:00}
messagebox(lcPub + ' converts to ' + ;
    transform(ltPub))
```

Summary

Craig Boyd has created very easy-to-use and inexpensive (free!) libraries we can add to our VFP applications to email and upload and download files. You'll love adding the new features to your applications that these libraries allow.

Doug Hennig is a partner with Stonefield Systems Group Inc. and Stonefield Software Inc. He is the author of the award-winning Stonefield Database Toolkit (SDT); the award-winning Stonefield Query; the MemberData Editor, Anchor Editor, and CursorAdapter and DataEnvironment builders that come with Microsoft Visual FoxPro; and the My namespace and updated Upsizing Wizard in Sedna.

Doug is co-author of "Making Sense of Sedna and SP2," the "What's New in Visual FoxPro" series (the latest being "What's New in Nine"), "Visual FoxPro

Best Practices For The Next Ten Years," and "The Hacker's Guide to Visual FoxPro 7.0." He was the technical editor of "The Hacker's Guide to Visual FoxPro 6.0" and "The Fundamentals." All of these books are from Hentzenwerke Publishing (http://www.hentzenwerke.com). He wrote over 100 articles in 10 years for FoxTalk and has written numerous articles in FoxPro Advisor, Advisor Guide to Visual FoxPro, and CoDe. He currently writes for FoxRockX (http://www.foxrockx.com).

Doug spoke at every Microsoft FoxPro Developers Conference (DevCon) starting in 1997 and at user groups and developer conferences all over the world. He is one of the organizers of the annual Southwest Fox conference (http://www.swfox.net). He is one of the administrators for the VFPX VFP community extensions Web site (http://vfpx.codeplex.com). He has been a Microsoft Most Valuable Professional (MVP) since 1996. Doug was awarded the 2006 FoxPro Community Lifetime Achievement Award (http://tinyurl.com/ygnk73h).