

Processing E-mail Orders Using Outlook and Access

Garry Robinson and Scott McManus



Garry Robinson and Scott McManus show how they accepted orders placed for their company's software through an online ordering system. With some Access code that links to Outlook, they process those orders automatically.

ONE of the most important components of running a Web site is the ability to accept orders from customers. Anyone who has successfully completed a secure Web site ordering system has my full admiration. It isn't easy to achieve. If you then had a large number of customers who used that ordering system successfully, you and your team are geniuses. For the rest of the Web sites in the world, the safest and easiest way to add an ordering system is to use one from a reputable third party.

This article shows how to read an order that arrives as an e-mail, store that information in your database, prepare an e-mail to inform your new customers that you've received their order, and then move that e-mail to another folder upon completion. The example that we've presented is designed to accept orders from DigiBuy.com, a third-party secure ordering Web site with many thousands of software developers as its clients. While this article is specific to DigiBuy.com, the code that we use here could be applied to any Web form or software system that produces text (e-mail) in a consistent format.

Preparing an e-mail

When we first thought about writing this article, we wondered how readers could get a sample e-mail order to try with our demonstration software. We solved this by building a sample order in text that you can e-mail to yourself. To generate such an e-mail, we wrote some code to create an instance of Outlook and then create a new Outlook e-mail:

```
Dim appOL As Outlook.Application
Dim testEmail As Outlook.MailItem

Set appOL = Outlook.Application
Set testEmail = appOL.CreateItem(olMailItem)
testE-mail.Subject = "Add your own e-mail address"
```

We generate a test order through a function that reads a text file. As in many previous articles, we use the

getDbPath function. This returns the relative path of the demonstration database, which is where we store the text file with the e-mail text body:

```
testE-mail.Body = TextFileToString_FX( _
    GetDBPath_FX & "MyFirstOrder.txt")
```

Finally, we display the e-mail so you can amend any details and add your e-mail address. We then clear the reference to the objects, as we no longer need them:

```
testEmail.Display

Set testEmail = Nothing
Set appOL = Nothing
```

If you want to test the system with multiple orders, please change the order number in the e-mail before you send it. Figure 1 shows the sample order that's generated. This has the same structure as an order e-mail that you'd receive from DigiBuy.com.

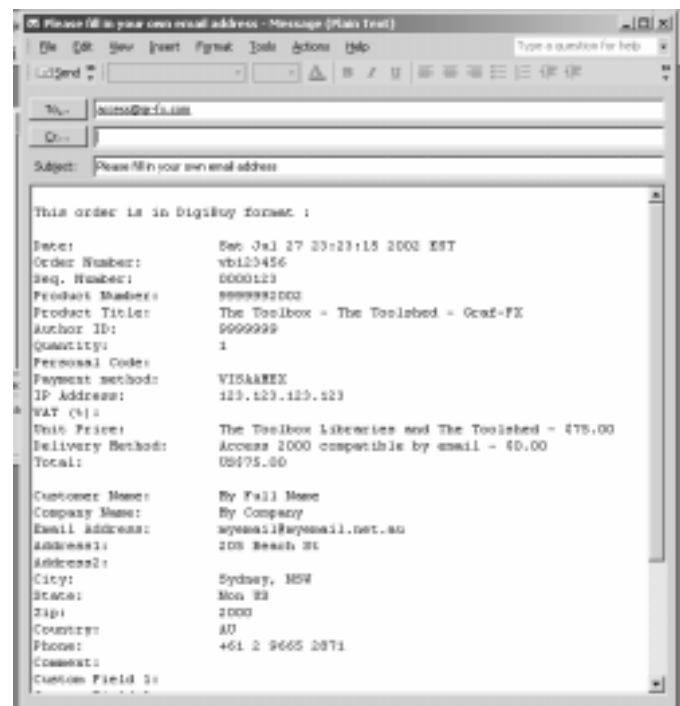


Figure 1. A sample e-mail order is generated for processing in the database.

Importing a text file

To make the body of the e-mail message, we've found the best way is to store the static information that we receive in text files. This seems to work well because it allows you to process the order even if you don't have access to Outlook and your Access database. The general function that follows can be used in any VBA program. It works by reading every line of the text file into a long string. When the end of the line is encountered, carriage return and line feed characters are added to the string.

```
Function TextFileToString_FX(fileName As String) _
    As String

Dim stemp, linesfromfile, nextline As String
Dim iFile As Integer

    TextFileToString_FX = ""
    On Error GoTo error_TextFileToString_FX

    iFile = FreeFile
    Open fileName For Input As iFile

    While Not EOF(1)

        Line Input #1, nextline
        linesfromfile = linesfromfile + nextline _
            + Chr(13) + Chr(10) Wend
    Close iFile

    TextFileToString_FX = linesfromfile

exit_TextFileToString_FX:

    Exit Function

error_TextFileToString_FX:

    MsgBox "Error opening file " & fileName _
        & " with " & Err.Description, vbCritical, _
        "Error Number " & Err.Number

End Function
```

If your order consists of multiple parts, it's easy to combine the various order e-mail files using the text files that are appropriate to the individual parts of the order.

Using Inbox rules

To start the process of reading the e-mails, the first step is to move the orders—as they arrive from the vendor—from your inbox into a special folder (we called it the Orders folder) for processing. We used the inbox rules wizard in Outlook to set up the rules to have mail moved to the Order folder when e-mails are downloaded from our ISP. To find out how to do this in Outlook, search Outlook Help for “rules wizard.”

In the following code, we have two constants that represent the names of the Outlook folders. You'll need to change these to your own folder names. Do *not* use subfolders, as the code for this in Outlook is quite tricky. The OrderTable and TipsList constants are used for storing the customer order details and the e-mail address for a newsletter, respectively:

```
Const OrdersInFolder = "_ORDERS"
Const OrdersDoneFolder = "_Orders Processing"
Const OrderTable = "SoftwareOrders"
Const tipsList = "TipsMailList"
```

Processing the order

To process the orders, we loop through all the e-mails in the Orders folder in Outlook. We then read the text in the body of the e-mails. The process starts by instantiating Outlook and gaining access to the two Outlook folders that we'll be processing:

```
Set dbs = CurrentDb

Set myolApp = CreateObject("Outlook.Application")
Set myNameSpace = myolApp.GetNamespace("MAPI")
Set myfolder = myNameSpace.Folders( _
    "Personal Folders").Folders(OrdersInFolder)
Set myNewFolder = myNameSpace.Folders( _
    "Personal Folders").Folders(OrdersDoneFolder)
```

Now we work through those Order e-mails and process them one at a time. We also need to open the table where we store the new customers' order details:

```
iMax = myfolder.Items.Count
If iMax = 0 Then
    MsgBox "Unfortunately there are no orders"
Else
    Set rstSoftOrders = dbs.OpenRecordset( _
        "SoftwareOrders", DB_OPEN_DYNASET)
    For iOrd = 1 To iMax
```

Once the e-mail is processed, we need to move it out of the Orders folder so that we don't process it again. As we move the e-mail message after it's processed, the next e-mail message moves to the top of the list. So, in our code, we always refer to item 1 in the Orders folder:

```
Set myItem = myfolder.Items(1)
```

Now we need to save the text of the order e-mail to a string variable called e-mailContents. Outlook provides access to the contents of the e-mail through the Items property Body. If you look back at Figure 1, you'll find the letter is broken into lines with the subject followed by a colon and a number of spaces. As these are always the same in every e-mail, we pass the body text to a function that extracts all the remaining text after the subject and the spaces (we'll explain this routine later):

```
e-mailContents = myItem.Body
UserName = ExtractToCR_FX(e-mailContents, _
    "Customer Name: ")
UnitPrice = ExtractToCR_FX(e-mailContents, _
    "Unit Price: ")
```

We display the information to the users and ask if they wish to proceed with the order. We then extract all the other fields in the e-mail body that we're going to store in Access. A portion of this code follows:

```
postIt = MsgBox(UserName & ": " & UnitPrice, _
    vbYesNoCancel, "Post The Following")
If postIt = vbYes Then

    On Error Resume Next
    rstSoftOrders.AddNew
    rstSoftOrders("Person") = UserName
    rstSoftOrders("OrderNumber") = ExtractToCR_FX( _
        e-mailContents, "Order Number: ")
    rstSoftOrders("SeqNumber") = ExtractToCR_FX( _
```

Continues on page 18

Orders Using Outlook...

Continued from page 8

```
e-mailContents, "Seq. Number:      ")
UserE-mail = ExtractToCR_FX(e-mailContents, _
"E-mail Address:      ")
rstSoftOrders("E-mail") = UserE-mail
rstSoftOrders("City") = ExtractToCR_FX( _
e-mailContents, "City:      ")
rstSoftOrders("State") = ExtractToCR_FX( _
e-mailContents, "State:      ")
```

Finally, you'll need to produce the outgoing acknowledgement e-mails that go back to the customer, based on their order. The details of generating the contents of the outgoing mail will differ with every application. To send the e-mail, we've used the venerable `SendObject` method:

```
UserAmountPaid = ExtractToCR_FX( _
e-mailContents, "Total:      US$")
rstSoftOrders("AmountPaid") = UserAmountPaid

On Error GoTo getOrdersDetails_error
rstSoftOrders.Update
On Error Resume Next

DoCmd.SendObject acSendNoObject, , acFormatTXT, _
UserE-mail, , , "The Toolbox from GR-FX", _
"Greetings " & UserName & ", " & vbCrLf & vbCrLf _
& TextFileToString_FX(GetDBPath_FX & "SA News.txt")
```

With all processing complete, we remove the Outlook e-mail item that we're processing from the Orders folder

and put it in the Orders Processed folder:

```
myItem.Move myNewfolder
MsgBox "Our Order for " & UserName & " " & _
UnitPrice & " .. " & UserE-mail & _
" >> has been moved to " & myNewfolder.Name
```

We can now process the next order.

In addition to storing the orders in a table, we add the users' names and e-mail addresses to a separate table so that they can receive e-mails about "items of interest" that relate to their purchases:

```
DoCmd.RunSQL "insert into " & tipsList & _
" values ('" & UserName & "','" & UserE-mail & "')
```

Processing the e-mail body

One important part of this software is the function that returns data from the e-mail body for a particular line of text in the e-mail. Our first step is to find the line that we want to process. We find the line by searching for a string constant in the e-mail. All text after that constant to the next carriage return is returned by the function `ExtractToCR_FX`. This method accepts the string to search and the identifying marker to look for. For an example of how this works, look again at Figure 1. That e-mail becomes a very long text string when extracted from the body of the e-mail. To retrieve the text for the Author ID, we use the following code:

Mailing Recordsets with Outlook

One of my clients needed to send data to a remote site on a regular basis. Rather than ship disks back and forth, we set up an e-mail system for moving data between locations. The benefits of e-mail were many: The data that we mailed showed up in Outlook's Sent Mail mailbox, effectively providing a log of activity; e-mail is stored and held until the recipient is ready to receive it; and the client could intervene in the process just by opening Outlook and deleting or resending e-mails.

One of the problems with sending data in e-mail is that you can only reliably send text. Fortunately, the ADO recordset's `Save` method allows you to convert recordset data into an XML text file with a single method call. Here's the code to retrieve a recordset and save it to an e-mail message, which requires that you use the stream object to bridge the gap between the mail item's `Body` property and the output

of the `Save` method:

```
Dim stm AS ADO.DB.Stream

Set rst = New ADO.DB.Recordset
Set stm = New ADO.DB.Stream

rst.Open "", strConnection, adOpenStatic, _
adLockBatchOptimistic, , adCmdFile
rec.Save stm, adPersistXML
stm.Position = 0
myItem.Body = stm.ReadText
```

When the e-mail is received, the recordset's `Open` method (with `adCmdFile` as the last parameter) loads that same XML text file back into a recordset for processing:

```
Dim rst As ADO.DB.Recordset
rst.Open myItem.Body, , , adCmdFile
```

—Peter Vogel

```
MyTextStr = ExtractToCR_FX(webstring, "Author ID: ")
```

This code would return "9999999" from the example in Figure 1 and put the result in the MyTextStr variable. The code for the function looks like this:

```
Public Function ExtractToCR_FX(textLine As Variant, _  
    FormItemReq As String) As String  
    Dim StartLine As Variant, EndLine As Variant  
    dim ExtractText As Variant  
  
    StartLine = InStr(textLine, FormItemReq)  
    If StartLine > 0 Then  
  
        StartLine = StartLine + Len(FormItemReq)  
        EndLine = InStr(StartLine, textLine, Chr(13))  
        ExtractText = Mid(textLine, StartLine, _  
            EndLine - StartLine)  
  
    End If  
    If Len(ExtractText) = 0 Then  
        ExtractText = " "  
    End If  
  
    ExtractToCR_FX = ExtractText  
  
End Function
```

The Download file's database

If you wish to try out this Outlook demonstration, you'll need a copy of either Outlook 2000 or Outlook 2002 on your computer. The database in the Download file (available at www.vb123.com/kb) is in Access 2000 format. It will require you to set up two folders in Outlook: one for storing the incoming message and one for holding the message after it's processed. The table fields used to store the customer details will require modifications to suit your system. The Download database requires the following external references:

- Microsoft Outlook 9.0 or 10.0 Object Library
- Microsoft DAO 3.51 Object Library

If you have Microsoft Outlook 97, you must program Outlook in VBScript rather than VBA (we won't cover that code in this article). Beware: If you upgrade to Outlook 2002 or 2000 version 2, some e-mail Automation won't work as it used to.

Using Microsoft Access and Outlook together can reduce the manual processing of order e-mails substantially. We know this because we started off by manually processing our e-mails. This could take up to 15 minutes to save customer details into tables and newsletter lists. Without software, it was also very difficult to explain to other staff members what to do when an e-mail arrived. Now we can process the orders in a couple of minutes once they arrive in the correct folder. As an added bonus, we can now demonstrate to our clients that we can program Microsoft Outlook. ▲

DOWNLOAD

[ORDOUT.ZIP at www.vb123.com/kb](http://www.vb123.com/kb)

Garry Robinson is the founder of GR-FX Pty Limited, a company based in Sydney, Australia. If you want to keep up-to-date with his latest postings on Access issues, visit his company's Web site at www.vb123.com or sign up for his Access e-mail newsletter by sending a blank e-mail to tips@gr-fx.com. The Web site features Access source code tools and resources. When Garry isn't sitting at a keyboard, he can be found viewing the real Outlook from one of Sydney's seaside cafes.

Scott McManus is the principal of Skandus. Scott's interests are in object-rich databases, XML, Outlook, and 3D objects in the mining industry. Outside the office, he enjoys archaeology, natural science, music, and the Mid North Coast of NSW.

Further Reading and Resources

From *Smart Access*:

- "From Access to Automated E-mail"—March 1999, by Garry Robinson
- "Taking Outlook and XML to Task"—July 2002, by Garry Robinson and Scott McManus
- "Get Your Access Data Into Outlook"—October 2000, by Helen Feddema
- "The Jet 4.0 Exchange/Outlook IISAM"—August 2000, by Michael Kaplan