Version: 5.0,6.0,7.0
Platform: Windows
Figures:
File:

REMOVing Email Addresses

Whil Hentzen

Sales campaigns conducted via email lists are a fact of life these days. If you've got a list of customers or potential customers, it makes good sense to make use of that list to contact them about products and services that they might be interested in. However, the issues that people change email addresses rather frequently, and that they hate receiving unwanted email are also facts of life. People who might have been interested at one point in time may no longer be interested, and you need to provide a way to remove them, as well as email addresses from which email has been bounced back, from your mailing list, and to do so efficiently and accurately. Here's one way.

I have a lot of email addresses – both for current and past customers as well as from potential customers – people who have downloaded from our site or otherwise asked for information about our products and services. I send out bulk email to this list on occasion for a variety of reasons – primarily to let them know about new books as well as news about GLGDW. Some of those email addresses go bad – people change jobs, email servers change, or accounts just get closed. Other times, people want an email address removed – either they've switched addresses, or perhaps they're simply no longer interested in what we're offering.

It's important to note that everyone in our database has had a legitimate business relationship of some sort with us – either they bought a copy of "TUFR" in 1994, or were on my "Mother of All User Group Lists" 1997, or downloaded files from a book of ours they bought in 2000. I've never bought lists from a third party – these are all people who have contacted us at some point in the past.

As a result, I feel more personally responsible that I don't annoy them if they don't want to receive email in the future. The simplest way is to flag those customers in my customer database so I don't mail to them again. Note that the remove requests HAVE to be removed – they've requested to be removed and I don't want to bother them again. The email that gets kicked back, due to an invalid address, on the other hand, doesn't really hurt anything, except that it clogs up things. Why use the bandwidth and time to send out if it's just going to get kicked back? So I'd like to flag those addresses as being bad as well.

In this article, I'm going to discuss how I handle remove requests. In the next one, I'll discuss handling bouncebacks.

How email is sent out

When I send out a bulk email, it has four important parts.

The first is the return address. I use a return address of remove@hentzenwerke.com, so that I can discretely identify remove requests and bouncebacks as coming in response to a bulk email, and not some other cause. With the proliferation of the Klez virus due to the security holes in Outlook, we get a lot of "third party redirects" that appear as bouncebacks to several of our published email aliases that have found their way into many people's address books. These miscreant email responses can usually be identified as such because they come back to one of those published aliases, not the 'remove' alias.

The second part is the first line in the body of the email. It tells the reader that remove instructions are found at the bottom of the email, like so:

```
(We don't want to bother you. See the bottom of this email for removal instructions.)
```

The third part of the email is the majority of the body of the email (other than the remove instructions at the end.)

*\\\

And, finally, the last important part of the email is the set of remove instructions themselves. Here is an example of what we put at the end of each email:

```
REMOVAL INSTRUCTIONS:

This email was sent to HERMAN@WERKE.COM because you've bought something from us, inquired
about something, or downloaded something from our website. If you'd rather not receive
email from us again, click REPLY and include the word "REMOVE" in the subject line to be
removed from all future Hentzenwerke mailings. Please make sure that the body of this
message is included in your reply as some of the addresses in our database may be
forwarded to other aliases that we can't trace back and remove. Thanks!
```

Note that I include the email address where the email was sent in the body, so that if the recipient follows instructions, I can find the address it was sent to and flag the correct email. This is important, because, as the remove instructions state themselves, the biggest challenge in handling remove requests is that people often forward one account to another – and in some cases, they even forget that they've done so. Then they reply from the second account, but since that second email address isn't in our database, an automated attempt to remove won't work, and the user's original email address is left unflagged.

People being people, this doesn't always work. Some people just don't follow instructions. They'll click reply, but not include "remove", so you're not sure what to do with it. Others will include "REMOVE" in the body, so it's not as easy to find in the slew of email bouncebacks that end up in the remove@hentzenwerke.com account.

And yet others will follow instructions, sort of. They'll include "REMOVE" in the subject line, but they'll delete the body. And there are always a few who reply from a different email address which is not in our database, so there's no direct way of matching them up with the address that the email was sent to. Finally, some email systems are set up to use a different email address as the 'from address' than what the user is used to thinking of as their email address. For example, you might think of your address as herman_werke@mycompany.com. But when you reply to an email, your "from" address shows up as herman_werke@email.mycompany.com. Each of these special situations needs to be taken care of by hand.

So this means that it's not a trivial matter to automate the process of handling remove requests. Yet, the majority of remove requests can be handled in a pseudo-automatic way. Here's what I've found works the best, and how I got there.

Using Automation to Remove

Since all of these remove requests end up in an Outlook PST, my first thought was to go through the PST, via Outlook Automation, and parse each message individually. The process would select at a remove request email, look up that email address up in my CUST database, and flag it to be removed. I figure I'd move each successfully parsed message to a different folder so I knew which messages had emails that were flagged.

I spent about a half day writing this code – well, actually, I spent about 3 hours and 45 minutes noodling the Automation help file and bugging friends, and 15 minutes writing the code. And that was all an awful experience.

- It's slow. Having to parse through a PST file, look for a remove message, and then work my way through it was a measurable process. I'm a FoxPro developer. I'm not used to seeing the grass grow under my record processing, right?
- It's awkward. There was a lot of work needed to find the right item, walk down the tree, open an item, work with it, and then move it to another folder. Intellectually it wasn't demanding; annoying is a more accurate description.
- Automation examples all use VB. Having to spelunk through the Automation help file and decipher the decidedly sparse examples was a trial – then converting the syntax from VB to Fox was even a worse chore. Yeah, it's close, but remember it only takes one semi-colon to cause a rocketship to explode.

So then I hit upon the idea of exporting the remove request emails from the PST to a DBF file… and I know how to handle DBFs in my sleep. This isn't a perfect solution, because the Outlook Export functionality leaves, er, a lot to be desired. For example, attachments can't be exported. Nor can dates. But instead of

having to spelunk through the VB automation code – and still run into many of the same problems, exporting the key items – the "from" email address and the body of the message – and then working with them inside Fox, was a much quicker solution.

Exporting to a DBF from Outlook

In order to export email messages, open up Outlook, fire up the Import and Export Wizard via the File | Import/Export menu option, and select Export to a File, as shown in Figure 1.

Figure 1. The first screen of Outlook's Import/Export Wizard.
08HentzenEmailRemove_01.TIF

After clicking Next, scroll down through the file types list and select Microsoft FoxPro as shown in Figure 2.

Figure 2. Select Microsoft FoxPro as the desired file type to export to.
08HentzenEmailRemove_02.TIF

After clicking Next again, you'll see your Personal Folders tree as shown in Figure 3. Double-click your way through to the folder that contains the messages you want to export (in my case, it's in a folder called "EmailRemove"), and click next.

Figure 3. Double-click through the treeview to the folder that contains the messages you want to export.
08HentzenEmailRemove_03.TIF

You'll be prompted for the filename where you want to export your messages. Use the text box and/or Browse button to navigate to the folder where you want the DBF created and to name the DBF itself. I name this table "EMAILREMOVE.DBF" (and when I export bouncebacks next month, you'll see I name that table "EMAILBAD.DBF.")

Figure 4. Identify the output DBF.
08HentzenEmailRemove_04.TIF

Now comes the frustrating part. You'll be asked to select which actions you want performed, although I've only ever seen one action in the "The following actions will be performed" listbox, as shown in Figure 5.

Figure 5. Select which actions you want performed.
08HentzenEmailRemove_05.TIF

If you click on the "Map Custom Fields" button, you'll get the screen shown in Figure 6. It leads you to believe that you can map all of your Outlook fields to corresponding fields in your FoxPro DBF, but that is not exactly right. The available fields in the left side of the dialog are only a subset of the attributes that belong to an email message. Attributes like Date Sent and Attachment aren't available for export. Considering this functionality is missing in Outlook 2000, the third incarnation of Outlook, I think this is pretty lame.

Figure 6. The Map Custom Fields leads you to believe you can map all of your Outlook fields to corresponding fields in your FoxPro DBF.
08HentzenEmailRemove_06.TIF

As a result, you can basically ignore the Map Custom Fields dialog and just go ahead and click the Finish button in Figure 5 to initiate the export itself. Once you do so, you'll get the flying papers progress dialog as shown in Figure 7,

Figure 7. The flying papers progress dialog shows you the progress you've made.

08HentzenEmailRemove_08.TIF

When the export process is complete, you'll have a DBF with the following structure:

```
EMAILREMOVE.DBF

 1  SUBJECT         Character             210
 2  BODY            Memo                    4
 3  FROMNAME        Character             210
 4  FROMADDRES      Character             210
 5  FROMTYPE        Character             210
 6  TONAME          Character             210
 7  TOADDRESS       Character             210
 8  TOTYPE          Character             210
 9  CCNAME          Character             210
10  CCADDRESS       Character             210
11  CCTYPE          Character             210
12  BCCNAME         Character             210
13  BCCADDRESS      Character             210
14  BCCTYPE         Character             210
15  BILLINGINF      Character             210
16  CATEGORIES      Character             210
17  IMPORTANCE      Character             210
18  MILEAGE         Character             210
19  SENSITIVIT      Character             210
```

Ask me why "BillingInf" and "Mileage" are included as part of the export process for an email, but date fields are not. I'll tell you that I don't know. It's interesting to open up the table in a browse window and scan through the records. A typical record is shown in Figure 8.

Figure 8. A typical email record after being exported.
08HentzenEmailRemove_09.TIF

We're interested in just a few fields in this table. **Body** contains the body of the email that was sent to remove@hentzenwerke.com. It may or may not have something in it. If it's empty, it may be empty, or it may contain a .NULL. value. **FromName** contains the real name of the person sending the email as they entered it into their email client. For some people, like AOL users, the FromName is the same as their email address. The **FromAddres** (note that there's only one 's' because this is a free table and so field names are limited to 10 characters) is the actual email address from which the email was sent.

We're going to use two other existing fields in this table for our own purposes, ignoring the info that's already in them. We're going to go through this table, and try to find each records' From Email address in our Customer table. If we're not successful, we're going to have to look for that record manually. We'll use **FromType** to mark this record in EMAILREMOVE.DBF as 'not found' so that we can come back to it later.

The Remove Instructions in the body of the email we sent out included the email address that the email was sent to. We'll pull that address out of the body and put it into a character field that's easier to manipulate when matching records against CUST. We'll use the CCADDRESS field to hold the actual email address – if there is one, that is. If the body is empty, we'll insert a semaphore into CCADDRESS indicating that the recipient didn't follow the remove instructions, and deleted the body when responding.

Finally, for the purposes of this article, I'm going to assume that our customer table has fields for first and last names (cNaF, cNaL), an email address (cEmail), a field for email status (cStatEmail) that contains "REMOVE" or "BAD" for those to whom we're not going to mail anymore, a field describing why and when the field was set to REMOVE or BAD (cWhyBad).

Matching and marking remove requests

OK, now onto the process.

The first step is to get 'good' email addresses set up in EMAILREMOVE so that they can easily be matched against the cEmail field in CUST.DBF. The first step is to replace the contents of the FROMADDRES field with the lowercase version, just to make sure that case issues don't trip us up later.

The second step is to strip out the email address that the bulk email was originally sent to and stuff it into the CCADDRESS field. Here's the code:

```
use EMAILREMOVE
replace fromaddres with lower(fromaddres) all
go top
scan
 if isnull(emailremove.body) or ;
    empty(alltrim(emailremove.body))
  * don't mess with this record if it's empty
  replace ccaddress with "No body"
 else
  * where the email address starts
  m.liStart = at("was sent to", ;
    lower(emailremove.body)) + 12
  * go through char by char until we find a space
  for m.li = m.liStart to m.liStart+100
   if substr(emailremove.body, m.li) = " "
    * m.li marks the end of the address
    m.liLast = m.li
    exit
   else
    * keep on going
   endif
  next
  m.liLength = m.liLast - m.liStart
  m.lcAddress = substr(emailremove.body, ;
    m.liStart, m.liLength)
  replace ccaddress with lower(m.lcAddress)
 endif
endscan
```

Not too bad, is it? Now we've got two fields in EMAILREMOVE.DBF that might contain an email address that we want to match in CUST.DBF.

We're first going to look for CCADDRESS, because that's the address we know we sent to. If CCADDRESS exists, we should always be able to find it in CUST.DBF. If we can't, it's likely because CCADDRESS is empty, because the recipient of our email didn't include the body in their remove request email.

So, the next piece of code rolls through each record in EMAILREMOVE, looking for the contents of CCADDRESS somewhere in the cEmail field in CUST.DBF. If the CCADDRESS data is found in CUST, then the FromType field in EMAILREMOVE is fileld with "FoundInCust", CUST.cStatEmail is replaced with "NEVER", and cWhyBad is replaced with the explanation of why the NEVER flag was set: RemoveReuqest CCYYMMDD"

```
* go through EMAILREMOVE and try to match
* CCADDRESS with CUST.cemail
m.liNumRec = reccount()
scan
 * look for the address in the body first,
 * cuz that's who we sent to
 if "@" $ ccaddress
  wait wind nowait "Looking for REMOVE email in CUST " ;
  + transform(m.liNumRec - recno())
  m.lcEmail = upper(alltrim(emailremove.ccaddress))
  * create a string explaining why this address is
  * flagged
  m.lcWhyBad = "RequestedRemove " + + dtos(date()) + ;
    "_" + strtran(time(), ":", "")
  select CUST
  if seek( m.lcEmail )
   * if we found the email, flag it
   if empty(cWhyBad)
    replace cWhyBad with m.lcWhyBad
    replace cstatemail with "NEVER"
    select EMAILREMOVE
    replace FromType with "FoundBodyInCust"
```

```
   else
    m.lcWhyBadOld = CUST.cWhyBad
    replace cWhyBad with alltrim(alltrim(m.lcWhyBad) ;
      + " " + alltrim(m.lcWhyBadOld))
    replace cstatemail with "NEVER"
    select EMAILREMOVE
    replace FromType with "FoundBodyInCust"
   endif && empty(cWhyBad)
  else
   * we didn't find the email in CUST, so flag
   * EMAILREMOVE.DBF with a note
   select EMAILREMOVE
   replace FromType with "NotFoundBodyInCust"
  endif && seek (m.lcEmail)
 else
  * there is no email address in ccaddress (cuz body
  * was null or empty)
  select EMAILREMOVE
  replace FromType with "NotFoundBodyInCust"
 endif && "@"$ccaddress
endscan
```

As FoxPro code goes, this is pretty trivial; it's the business logic more than sophisticated programming that I'm interested in here. (I haven't tested it, but I bet all of this code would work in FoxPro 2.0.)

Now, suppose we can't find CCADDRESS in CUST.cEmail? Remember, the recipient of our email might have deleted the body before replying, so all we have to go on is the FromAddres value, which may or may not be the address to which the bulk email was sent to. A similar batch of code will do the trick, with the exception that we're only going to use FromAddres if we didn't have a match on the CCADDRES value, and the two values are different.

```
* if we didn't find the ccaddress, and the
* fromaddres is different than the ccaddress,
* look for the fromaddres
select EMAILREMOVE
go top
scan
 wait window nowait "Looking for REMOVE email in CUST " ;
 + transform(m.liNumRec - recno())
 if FromType <> "Found" and ;
    alltrim(emailremove.fromaddres) <> ;
    alltrim(emailremove.ccaddress)
  m.lcEmail = upper(alltrim(emailremove.fromaddres))
  m.lcWhyBad = "RequestedRemove " + + dtos(date()) + ;
    "_" + strtran(time(), ":", "")
  select CUST
  if seek( m.lcEmail )
   * if we found the email, flag it
   if empty(cWhyBad)
    replace cWhyBad with m.lcWhyBad
    replace cstatemail with "NEVER"
    select EMAILREMOVE
    replace FromType with "FoundFromInCust"
   else
    m.lcWhyBadOld = CUST.cWhyBad
    replace cWhyBad with alltrim(alltrim(m.lcWhyBad) ;
      + " " + alltrim(m.lcWhyBadOld))
    replace cstatemail with "NEVER"
    select EMAILREMOVE
    replace FromType with "FoundFromInCust"
   endif && empty(cWhyBad)
  else
   * we didn't find the email in CUST, so flag
   * EMAILREMOVE.DBF with a note
   select EMAILREMOVE
   replace FromType with alltrim(FromType) ;
     + iif(!empty(FromType), "", "") ;
     + "NotFoundFromInCust"
  endif && seek (m.lcEmail)
```

```
 endif && alltrim(emailremove.fromaddres) <>
endscan
wait clear
```

In my tests, these two passes through EMAILREMOVE find most of the remove requests – usually over 90%. But there's still that 10% that have to be handled – and they do have to be handled, because they're not going to realize that they didn't follow instructions (and, once in a while, they just don't follow instructions to be ornery, not quite understanding that they're kind of subverting their own purpose), but they'll still be bugged if they continue getting email.

So it's time to roll up your sleeves and do the final flagging manually. First, set a filter on EMAILREMOVE so that you only see FromType = "NotFound". Next, use the value of what looks to be the person's last name in the FromName field to search in CUST – you'll often find that the email is similar enough that you can tell that it's that person. Finally, for the few that can't be found that way, spelunk via the domain name. For example, if you're looking for john_doe@mycompany.com, but you can't find DOE in the last name field or the email address, trying searching for the "mycompany" domain. I've often found a record like john_l_doe@mycompany.com, or jdoe@email.mycompany.com.

Simply going through these two steps takes care of the majority of the remaining remove requests. The remaining folks? Well, they're just going to be mailed to again and again until they can follow instructions properly.

Finally, copy EMAILREMOVE to a database archive directory, in case you get an irate person who complains you haven't gotten rid of them – you can hunt them down in your data and see what's going on. Most people, no matter how unhappy at first, if they see that you've made your best efforts to flag their record, will settle down. And the few who don't, well, there's one in every crowd and it's not worth losing sleep over them.

The beauty of this is that each time you do a bulk email, the number of remove requests will go down. Sure, you'll likely always get a few, but after the first few times, the majority of people who don't want to hear from you again will have already answered, and you'll have flagged them, so they don't more email from you.

Next month, we'll take a look at handling the bouncebacks, a much trickier issue.

Whil Hentzen is editor of FoxTalk.