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

Parsing Bad Email Addresses

Whil Hentzen

This month I'll discuss the second half of dealing with the remnants of bulk emailing – examining why email is bounced back to you, and how to deal with it.

As I mentioned last month, when you do any type of mass emailing, you're going to get some bounced back. There are a variety of reasons that this will happen. Most often it's because the address is no longer valid. Other reasons include an email server that is down, the recipients' mailbox is full, or variety of Internet errors (too many hops). Heck, it's even possible that the address is bad (the user mistyped their email address when entering it into your database.) And once in a while, it's because email from your server is being refused. In all of these cases, the email you sent is going to come bouncing back to you – specifically, to the address that sent them.

And, over the last few months, I've also seen a number of bounce backs due to the Klez virus and its variants. Klez is nefarious because it uses a two pronged approach to spreading. First, as many other viruses do, it takes advantage of the insecure architecture in Outlook by sending itself to people in an infected person's address book. Second, though, it forges the 'from header' in the email that it's propagating, so that it appears that the email comes from someone else in the address book, not from the infected person. In other words, suppose Al's computer is infected, and he has Barb and Carla in his Outlook address book.

When Klez attacks, it sends an email to Barb, but instead of using Al as the "From" address, it uses Carla. This is rather frustrating, of course, because Barb gets mad at Carla for sending a virus-laden email, not at Al, who is the justifiable target.

The reason we get bouncebacks is that people occasionally put our 'books@hentzenwerke.com' email address in their address book by mistake. Then when they get hit by Klez, they send out email to other people, but with "books@hentzenwerke.com" as the from address, not the actual originator of the email. Some of those address aren't any good, and so those emails are bounced back (or sometimes refused), and the bounce back message is sent to the 'from address' – books@hentzenwerke.com.

The bottom line is that when dealing with bounce backs, you have to wade through a lot of garbage and that garbage is in a lot of different formats.

You should have a separate account set up to receive bounce backs and other responses, and if you're running a large amount of email, you may want to set up more than one account. For purposes of this article, I'll assume that you've set up just one email address solely for your bulk email, and can thus segregate all of your bounce back messages from your other email. Still, as I said, you'll have a lot of garbage in that account, and it comes in a wide variety of formats.

Why is this? Bounce backs are processed by the mail server program of the recipient, and different mail servers handle bounce backs in different ways. A mail server is just a software program, and the message returned to the sender of an email that the email server receives is just part of that program's code. The bounce back message from **sendmail** will be different than that of Gordano's NTMail, which will again be different than Lotus Notes.

Furthermore, the message from a particular program may vary according to how it was installed and configured. Bounce back messages from Joe's ISP using SuperMail could be different than bounce back messages from MegaCom which is also using SuperMail. And, finally, the content and format of the message may vary according to the reason that the message was refused. Still, there are bound to be a fair number of similarities – there are just so many ways to tell someone "your message was refused by our server."

In order to handle each of these types of messages, it'll be necessary to know what the formats are. How to determine them?

As with last month's article on Remove Requests, I first thought I'd go through my Outlook PST via Office Automation and parse message by message. We already know why that was a pain. So, again, I

exported the bounce back messages to a DBF and worked with that, just as in last month's article. When finished, I had a DBF and FPT named EmailBad. A sample message is shown in Figure 1.

Figure 1. A typical email record after being exported.
09HentzenEmailRemove_01.TIF

I've analyzed tens of thousands of bounce backs over the years, and fortunately found a fair number of similarities. There were two groups of bounce backs.

The first type contained all the info we needed in the body of the bounce back message itself. All of the necessary info is in From, Subject, and Body for many records.

The second type didn't include the email address of the intended recipient, but included the original email as an attachment. That's going to be considerably more trouble, so for the purposes of this article, I'm just going to concentrate on those emails where I could succeed just using the contents of body of the bounce back email. I've found that between 60% and 80% of the bounce backs we get can be successfully handled without having to deal with the attachment.

The same subject lines, and, correspondingly, similar bodies, kept showing up over and over again. Thus, common subject lines and bodies mean I can parse those messages automatically, find the recipient's email address, look it up in my CUST database, and set a flag indicating not to email them again, and when and why that flag was set.

I dug out bounce backs that dated over the past couple of years, did a SELECT DISTINCT on Subject, and got the following results:

% of records	Subject line
69.6	Mail System Error – Returned Mail
6.9	Undeliverable Message
5.5	Undeliverable: <subject line>
3.6	Failure notice
3.1	Returned mail: User unknown
1.7	Returned mail: see transcript for details
1.6	Subject line contains “@” (see next)
1.4	delivery failure: <address> lines)
1.4	Delivery Status Notification (Failure)
0.9	Mail delivery failed: returning message to sender
0.9	Undelivered Mail Returned to Sender

Other messages include “Returned mail: Service unavailable”, “Warning: could not send message for past 4 hours”, and “Returned mail: Mailbox full”. A variety of “undeliverable” type messages also show up on occasion. Obviously, handling just a few common types of subject lines will take care of most of the bounce backs. So how do we grab addresses from here?

The idea is to go through EMAILBAD.DBF, parse through the subject and the body, looking for an email address. After looking at hundreds of these manually, it turns out that the first email address in the body is virtually always the address to which the email was sent. Once that address has been snared, try to match that address against an address in CUST.DBF. If found, set a flag in CUST.DBF that the address is bad, and indicate the reason - undeliverable, addressee unknown, whatever. Then, flag EMAILBAD that the address was found (or not found). Finally, once the process is complete, I archive the file in case I need to refer to it again.

There are three parts to this program. The first routine identifies the type of message being examined, and sets up some parameters that indicate where the address might be. These parameters include whether the string in which to look (either the subject line or the body), where to start looking (in many cases, the email address is always found after a particular string of text, such as “The following recipient was not found:”), and what the delimiters for the address are.

For example, a message with the subject line “Mail System Error – Returned Mail” often has text like the following in the body:

```
Each of the following recipients was rejected by a remote mail server.
The reasons given by the server are included to help you determine why
```

```
each recipient was rejected.

    Recipient: <george.jungle@usa.net>
    Reason:    <george.jungle @usa.net>... User not known
```

Thus, the string that's going to be searched is that whole block of text. We don't have to start looking at "Each of the following", though, because the email address is always found after "Recipient:", so that string is a second parameter. And the email address is always surrounded by "<" and ">" delimiters. So those four pieces of data will be sent to a second routine which will parse out "george.jungle@usa.net" from the body of the message.

The second routine finds the address given those parameters and stuffs the resulting value into an unused field in EMAILBAD.DBF (CCADDRESS, the same field as in the EMAILREMOVE program from last month.)

And the third routine then scans through the entire EMAILBAD table, looks for the address in CUST and set flags as appropriate. Why don't I intertwine the second and third routines? I guess I could, but as I was doing the testing for these routines, I needed to take intermediate snapshots of the data in order to make sure I was grabbing the right information. It was more efficient to examine the EMAILBAD table after parsing the address but before looking for the addresses in CUST.DBF.

Here are the case statements for the most common subject lines:
The first case doesn't require us to parse the body of the message because the subject line contains the email address for which delivery failed.

```
case "@" $ m.lcSubject
 * don't parse m.lcBody because we just have to look
 * at the Subject line!
 *" Subject line contains "@"
 * (1.4% had DELIVERY FAILURE: <address> lines)"
 m.lcStringToLookIn = m.lcSubject
 m.lcStringPre = ""
 m.lcStringDelim1 = "("
 m.lcStringDelim2 = ")"
 m.lcReason = "@ in subject line"
```

The second case is from CompuServe and contains static, easily identifable text in both the subject line and the body.

```
case m.lcSubject = "Undeliverable Message" ;
  and m.lcFromName = "CompuServe Postmaster"
 m.liPositionEnd = at("Sender: remove@hentzenwerke.com",;
   m.lcBody) + 32
 m.lcReason = "Undel/CompuServe PM/parsebody"
 m.lcStringToLookIn = substr(m.lcBody, m.liPositionEnd)
 m.lcStringPre = "for"
 m.lcStringDelim1 = "<"
 m.lcStringDelim2 = ">"
```

The third case simply contains the email address in the body.

```
case m.lcSubject = "Returned mail:"
  m.lcReason = ["Returned mail:"]
  m.lcStringToLookIn = m.lcBody
  m.lcStringPre = ""
  m.lcStringDelim1 = "<"
  m.lcStringDelim2 = ">"
```

The fourth case actually looks for several types of subject lines that all contain the string "Undeliver" and have the string "To:" in the body. Messages of these types include "Undeliverable" and "Undelivered".

```
case m.lcSubject = "Undeliver:"
  m.lcStringPre = "To:"
  m.lcStringToLookIn = m.lcBody
  m.lcStringDelim1 = " "
  m.lcStringDelim2 = "chr(13) "
  m.lcReason = [Undeliverable]
```

The fifth case is unusual in that the subject line starts out in lower case, while every other common subject line is properly capitalized.

```
 case m.lcSubject = "failure notice"
  m.lcStringPre = ""
  m.lcStringToLookIn = m.lcBody
  m.lcStringDelim1 = "<"
  m.lcStringDelim2 = ">"
  m.lcReason = [failure notice]
```

Finally, the last case has two pieces to it. In both cases, the subject line reads “Mail System Error” but the contents of the body are formatted one of two ways. This case handles both.

```
case m.lcSubject = "Mail System Error - Returned Mail"
 do case
 case "Recipient: <" $ m.lcbody and ;
    "Reason:    " $ m.lcbody
  m.lcReason = ["Recipient: <" $ m.lcbody and ;
    "Reason:    " $ m.lcbody]
  m.lcStringToLookIn = m.lcBody
  m.lcStringPre = "Recipient:"
  m.lcStringDelim1 = "<"
  m.lcStringDelim2 = ">"
 case "The following recipients did not receive ;
    this message:" $ m.lcbody
  m.lcReason = ["The following recipients did not ;
    receive this message:" $ m.lcbody]
  m.lcStringToLookIn = m.lcBody
  m.lcStringPre = "this message:"
  m.lcStringDelim1 = "<"
  m.lcStringDelim2 = ">"
 othe
  m.lcReason = "Mail System Error but otherwise CASE"
 endcase
```

Now that the parameters have been identified for this email message, I stuff the values into unused fields in EMAILBAD, like so:

```
replace BCCNAME with m.lcStringPre
replace BCCADDRESS with m.lcStringDelim1
replace BCCTYPE with m.lcStringDelim2
replace CCTYPE with m.lcReason
```

OK, now that I've run a message through these cases, I've got values in m.lcReason, m.lcStringToLookIn, m.lcStringPre, and the two delimiters. I'll send these values through the following code updates EMAILBAD and calls a function that looks for an email address and returns it if found.

```
if m.lcReason <> "--"
 m.lcAddress = l_GetAddress(m.lcStringToLookIn, ;
    m.lcStringPre, m.lcStringDelim1, m.lcStringDelim2)
 if empty(m.lcAddress)
   * didn't find an address in cBody or cSubject
   * of EMAILBADADDR.DBF
   replace CCADDRESS with "Not found"
 else
   replace CCADDRESS with m.lcAddress
 endif
else
 replace CCADDRESS with "-- didn't test --"
endif
```

I've found these seven cases take care of about 85% of the bouncebacks. Of the rest, about 5% are messages we don't want to mess with – mailbox full or system busy type of messages. We don't want to

delete these messages because the addresses are good – there's just something in the way that might get cleared up by the next time we want to mail again.

The l_GetAddress function is the second routine I mentioned - a UDF that takes four parameters and searches for an "@" sign. It searches the string, m.lcStringToLookIn, starting with the position where m.lcStringPre is found. The function uses m.lcStringDelim1 and m.lcStringDelim2 to know where to look.

The UDF itself isn't reproduced here but is included in the download file for this article.

Once EMAILBAD is populated with addresses in the CCADDRESS field, I used a process similar to last month's REMOVE routine to flag the CUST table. The routine also inserts the success or failure status of finding the address in CUST into EMAILBAD. The final step in this process is to archive EMAILBAD off and get rid of the email from the PST file.

Unlike the Remove process described last month, this process won't be perfect, but then again, it doesn't have to be perfect. We're just trying to make our bulk mailing more efficient by getting rid of addresses that don't work anymore. There's no benefit to spending a lot of time getting each and every last email. I'm going to spend the least amount of time necessary to pick the low hanging fruit,

As with many things, it's a case of diminishing returns – I want to get the most addresses with the least amount of work. If I miss a few, that's OK. Unlike messing up a remove request, a bounceback isn't upsetting anyone, and the extra effort our mail server goes through sending out 2or 4% more email isn't that big a deal. Your mileage, of course, may vary.

Whil Hentzen is editor of FoxTalk.