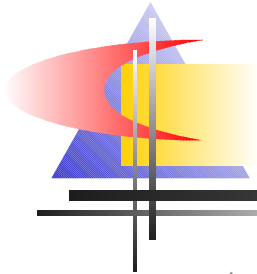


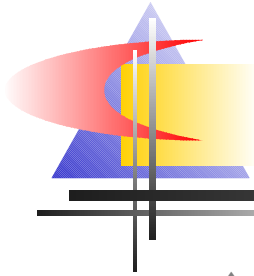
Real-world email handling in python

Anders Hammarquist



Real-world email handling in python

- ♦ A review of how the world bends the standards and how to feed it into Python and survive
- ♦ By now CAPS has seen tens, possibly hundreds, of thousands of email messages
- ♦ This is the story of what we've done to not fall over when something strange comes in



Background (CAPS and email)

- ♦ Email is a primary means of communicating with customers. It must not break!
- ♦ Messages must be stored in a way that is compatible with the rest of CAPS, and generated in a way that is understood by the MUAs out there.
- ♦ All text in CAPS is unicode
- ♦ We will explore all the things that can go wrong when trying to communicate using rfc-822 and friends.



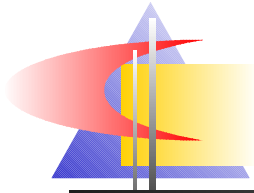
So what's the problem?

- ♦ MIME!
- ♦ Almost all problems we've seen are related to MIME
- ♦ If you stick with 7-bit us-ascii text/plain things work just fine



Encodings (fitting more bits in 7)

- ♦ Three types
 - Body: räksmörgås
 - Unstructured header fields:
`=?iso-8859-1?q?r=E4ksm=F6rg=E5s?='`
 - Structured header fields:
`title*=iso-8859-1'sv-se'r%E4ksm%F6rg%E5s`
- ♦ `email.Message` deals with Body and structured header fields, apply `email.Header.decode_header()` for unstructured headers.



Decoding the body

- ♦ 7bit and 8bit are easy and straight-forward
 - multipart/* and message/* must be
 - sometimes mislabeled
- ♦ quoted-printable and base64 are sometimes broken, often MIME-unaware software that adds a footer, or spam or viruses with broken base64
 - Found on a Sourceforge mailinglist
Content-Type: text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: quoted-printable
...
http://ads.osdn.com/?ad_id=1470&alloc_id=3638&op=click
- ♦ Python 2.3 deals with broken encodings (2.2 didn't)



Decoding the body

- ♦ email.Parser may fail for various MIME formatting problems

```
p = email.Parser.Parser()  
m = p.parse(fp, headersonly=True)  
bodystart = fp.tell()  
try:  
    p._parsebody(m, fp)  
except:  
    fp.seek(bodystart)  
    m.set_type('text/plain')  
    m.set_payload(fp.readline())
```



Decoding the header

- ♦ No 8-bit data (but it happens)
- ♦ Two types of encodings:
 - RFC-2047-style for unstructured header fields (i.e. text and comments)
 - RFC-2231-style for structured header fields (parameter values of MIME headers)
- ♦ RFC-2231 isn't well supported, so you will see RFC-2047-style encodings in structured headers.



Filenames (RFC-2231 vs MS)

- Outlook sends thinks like
filename="=?ISO-8859-1?Q?r=E4ksm=F6rg=E5s?=" and knows nothing about RFC-2231. So how do we make Outlook-users happy?
 - Two "filename" parameters
 - Content-Type: name= (deprecated)
 - Content-Disposition: filename=
 - Put RFC-2047 in "name" and 2231 in "filename"



Filenames (RFC-2231 vs MS)

- ♦ To parse the incoming header
 - if `Message.get_param()` returns something RFC-2047-like decode that with `email.Header`
 - otherwise use whatever came back (plain or 2231)
- ♦ `Header.decode_header()` on the entire header gives `filename=" räksmörgås "` (note spaces)



Character sets

- ♦ No matter what encoding, all email.* gives you after decoding is an 8-bit byte stream
 - 'r\xC3\xA4ksm\xC3\xB6rg\xC3\xA5s' or 'r{k|sm|rg}s'
- ♦ To do more, find the character set, convert
- ♦ Be prepared for Python not knowing the character set (SEN-850200-B is unknown, which is what r{k|sm|rg}s is)
- ♦ Possibly bail with ISO-8859-1 (character values stay the same in unicode)



Newlines (CRLF or LF?)

- ♦ RFC-2822 says line endings are CRLF
- ♦ Python tends to use LF
- ♦ so we must convert...
- ♦ Twisted SMTP and smtplib convert "raw" data
- ♦ But what if our character set isn't 7bit?



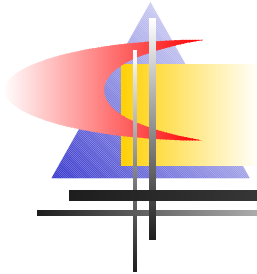
Newlines in 8bit data

- ♦ RFC-2045 says base64-encoded text must use CRLF (and sendmail will remove plain-LF if it converts base64 to 8bit)
- ♦ Python's base64 encoder doesn't convert so how about converting to CRLF before constructing an email.Message?
- ♦ It works great for character sets that do get encoded, but not so great for us-ascii...



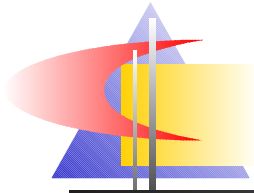
Newlines...

- ♦ So CRLF works if the character set need converting (it works with quoted printable too), but breaks with us-ascii.
- ♦ Solution:
 - Convert everything but us-ascii to CRLF
- ♦ Better fix (in the library):
 - Get the email.Charset converters to treat data as non-binary



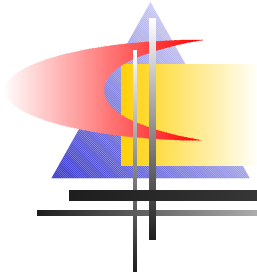
Incoming newlines

- ♦ Be prepared for CRLF even if you think it's all LF
- ♦ base64-encoded parts will use CRLF



Summary

- ♦ Know your RFCs
 - 2822 (Internet message format)
 - 2045-2049 (MIME)
 - 2231 (Parameter values in non-ascii)
- ♦ Be prepared that messages violate **EVERYTHING**
- ♦ Still possible to build a fairly robust message handling system



Questions

