# Creating and Sending Emails with File Attachments in Python

## *September 04, 2010*

*This is a good example of how to create and send emails with file attachments in Python.*

*It assumes, however, that every attached file is binary, and uses a generic `application/octet-stream` mimetype for each.*

*It also makes encoding assumptions about the strings passed to the Subject line, recipient email addresses, and the message body text which may be incorrect in actual use.*

*This updated version addresses those potential problems, no pun intended.*

***Attached File Mimetype***

*Instead of assuming `application/octet-stream`, we can use the python-magic module (available here, here, or by `apt-get install python-magic` on Debian and Ubuntu systems) to determine it explicitly:*

```
MIME_MAGIC = None
try:
    import magic
    MIME_MAGIC = magic.open(magic.MAGIC_MIME)
    MIME_MAGIC.load()
except ImportError:
    pass

def get_file_mimetype (filename):
    """Return the mimetype string for this file"""
    result = None
    if MIME_MAGIC:
        try:
            result = MIME_MAGIC.file(filename)
        except (IOError):
            pass
    return result
```

*The get_file_mimetype() function returns a string in the form `'Content-Type major type/Content-Type minor type'`, e.g. `'text/plain charset=us-ascii'`, `'application/pdf'`, etc.*

*Now we can change the loop that attaches files to the message from this:*

```
    for file in files:
        part = MIMEBase('application', "octet-stream")
        part.set_payload( open(file,"rb").read() )
        Encoders.encode_base64(part)
        part.add_header('Content-Disposition', 'attachment; filename="%s"'
                       % os.path.basename(file))
        msg.attach(part)
```

*To this, below. Note that since the mimetype can be text, we change the file read flag from "rb" (binary) to just "r", as necessary.*

```
    for file in files:

        file_read_flags = "rb"
        try:
            mimestring = get_file_mimetype(file)
            if mimestring.startswith('text'):
                file_read_flags = "r"
            mimestring_parts = mimestring.split('/')
            part = MIMEBase(mimestring_parts[0], mimestring_parts[1])
        except AttributeError, IndexError:
            # cannot determine the mimetype so use the generic 'application/octet-stream'
            part = MIMEBase('application', 'octet-stream')

        part.set_payload( open(file, file_read_flags).read() )
        Encoders.encode_base64(part)
        part.add_header('Content-Disposition', 'attachment; filename="%s"'
                       % os.path.basename(file))
        msg.attach(part)
```

***Subject Encoding***

*The original version used a simple assignment to define the Subject line:*

```
    msg['Subject'] = subject
```

*But this limits the Subject line to 7-bit ASCII characters only. For foreign language support and other encodings, it's better to use the* `email.Header` *package, which requires an additional import:*

```
from email.Header import Header
```

*The Subject line assignment changes to:*

```
    # always pass Unicode strings to Header, otherwise it will use RFC 2047 encoding even on plain ASCII str
ings
    msg['Subject'] = Header(to_unicode(subject), 'iso-8859-1')
```

*Where the to_unicode() function is defined as:*

```
def to_unicode (s):
    """Convert the given byte string to unicode, using the standard encoding,
    unless it's already encoded that way"""
    if s:
        if isinstance(s, unicode):
            return s
        else:
            return unicode(s, 'utf-8')
```

***Email Address Encoding***

*Unlike the Subject line, all email addresses must be ascii, so instead of defining the recipient list like this:*

```
    msg['To'] = COMMASPACE.join(to)
```

*We should map an explicit ascii encoding function over each email address, like this:*

```
    msg['To'] = COMMASPACE.join(map(lambda x: x.encode('ascii'), to))
```

***Body Text Encoding*** *Finally, the message body text, regardless of whether or not it's plain text, html, or both, must be unicode. So we go from this:*

```
    msg.attach( MIMEText(text) )
```

*To this:*

```
    msg.attach(MIMEText(to_bytestring(text), 'plain', 'utf-8'))
```

*If we want an html message body, we would do this:*

```
    msg.attach(MIMEText(to_bytestring(html_text), 'html', 'utf-8'))
```

*Actually, if you are going to use html in email messages at all, the best practice is to provide both a plain text and an html equivalent together, like this:*

```
    msg.attach(MIMEText(to_bytestring(text), 'plain', 'utf-8'))
    msg.attach(MIMEText(to_bytestring(html_text), 'html', 'utf-8'))
```

*In all the examples above, the to_bytestring() function is defined as:*

```
def to_bytestring (s):
    """Convert the given unicode string to a bytestring, using the standard encoding,
    unless it's already a bytestring"""
    if s:
        if isinstance(s, str):
            return s
        else:
            return s.encode('utf-8')
```

***A Complete Example***

*Putting it all together, this function lets you send the same email to multiple recipients, with optional files (binary or text) as attachments, and an optional message body in html.*

*It also allows you to define the "Reply-To" header of the message as a email address different from the one used to send the message.*

```
def send(sender, subject, recipient_list=[], text, html=None, files=[], replyto=None):
    """Send a message to the given recipient list, with the optionally attached files"""
    msg = MIMEMultipart('alternative')
    msg['From'] = sender.encode('ascii')
    # make sure email addresses do not contain non-ASCII characters
    msg['To'] = COMMASPACE.join(map(lambda x: x.encode('ascii'), recipient_list))
    if replyto:
        # make sure email addresses do not contain non-ASCII characters
        msg['Reply-To'] = replyto.encode('ascii')
    msg['Date'] = formatdate(localtime=True)

    # always pass Unicode strings to Header, otherwise it will use RFC 2047 encoding even on plain ASCII str
ings
    msg['Subject'] = Header(to_unicode(subject), 'iso-8859-1')

    # always use Unicode for the body text, both plain and html content types
    msg.attach(MIMEText(to_bytestring(text), 'plain', 'utf-8'))
    if html:
        msg.attach(MIMEText(to_bytestring(html), 'html', 'utf-8'))

    for file in files:

        file_read_flags = "rb"
        try:
            mimestring = get_file_mimetype(file)
            if mimestring.startswith('text'):
                file_read_flags = "r"
            mimestring_parts = mimestring.split('/')
            part = MIMEBase(mimestring_parts[0], mimestring_parts[1])
        except AttributeError, IndexError:
            # cannot determine the mimetype so use the generic 'application/octet-stream'
            part = MIMEBase('application', 'octet-stream')

        part.set_payload( open(file, file_read_flags).read() )
        Encoders.encode_base64(part)
        part.add_header('Content-Disposition', 'attachment; filename="%s"'
                        % os.path.basename(file))
        msg.attach(part)

    smtp = smtplib.SMTP(mail_server)
    smtp.sendmail(sender, recipient_list, msg.as_string() )
    smtp.close()
```

---