bodatools Documentation

Release 0.1.0

Emory University Libraries

June 08, 2012

CONTENTS

bodatools is a collection of Python code with tools for use with born-digital archival materials. Currently, this consist of:

- **bodatools.binfile**: provides the capability to map arbitrary binary data into a read-only Python object
  - includes BinaryStructure definitions for Eudora and Outlook Express 4.5 (Macintosh) email formats
- A command-line script:
  - `export-email` - export Outlook Express 4.5 (Macintosh) messages to individual files per email/attachment

bodatools was created by the Digital Programs and Systems Software Team of Emory University Libraries.

CHAPTER

ONE

CONTENTS

1.1 Scripts

Command-line scripts for use with born-digital archival materials.

1.1.1 export-email

export-email is a command-line script for exporting email content from binary, proprietary folder formats into individual email messages and attachment files, to allow individual email messages to be scrutinized and archived.

Currently, the only supported email format is Outlook Express 4.5 for Macintosh, but we expect to add support for other formats as we encounter them.pp

Example usage:

```
$ export-emails "Internet Mail" email-output
```

where **Internet Mail** is the directory that contains Outlook Express folders such as Inbox, Outbox, and Sent Mail, and **email-output** is a folder where the exported messages should be saved. On our hardware, the Outlook Express directory structure looks like this:

```
Internet
+- Internet Applications
   +- Outlook Express 4.5 Folder
      +- OE User(s)
         +- Account Name 1
         |  +- Internet Mail
         |     +- Deleted Messages
         |     +- Drafts
         |     +- Inbox
         |     +- Outbox
         |     +- Sent Mail
         +- Account Name 2
         |  +- Internet Mail
         |     +- Deleted Messages
         |     +- Drafts
         |     +- Inbox
         |     +- Outbox
         |     +- Sent Mail
```

If you with to export email messages for multiple accounts, you will need to run `export-email` once for each account, giving it the appropriate `Internet Mail` folder name.

output

As it runs, `export-email` will report on the Mail folders it identifies, the number of messages expected and the number of messages and attachments exported.

The script generates individual text files for email messages in each folder. The generated filenames include the folder name, message date, who the message was from (or who the message was sent to for the Sent Mail folder), and the subject. If an email message includes attachments with a filename, `export-email` will create a file in a `_parts` directory matching the output file name for the email message.

Note: In some cases, the folder `Index` file may reference deleted messages; these are currently skipped for output, but if any are found, a count of skipped deleted messages will be reported when the script runs. In some cases, there may also be content sections in the `Mail` data file which are not referenced by the folder `Index` (i.e., when the index and data files were not completely synchronized); these cannot exported, but a summary of the number of skipped sections is included in the script output in case further investigation is needed.

1.2 `bodatools.binfile` – Map binary data to Python objects

Map binary data on-disk to read-only Python objects.

This module facilitates exposing stored binary data using common Pythonic idioms. Fields in relocatable binary objects map to Python attributes using a priori knowledge about how the binary structure is organized. This is akin to the standard `struct` module, but with some slightly different use cases. `struct`, for instance, offers a more terse syntax, which is handy for certain simple structures. `struct` is also a bit faster since it's implemented in C. This module's more verbose `BinaryStructure` definitions give it a few advantages over `struct`, though:

- This module allows users to define their own field types, where `struct` field types are basically inextensible.
- The object-based nature of `BinaryStructure` makes it easy to add non-structural properties and methods to subclasses, which would require a bit of reimplementing and wrapping from a `struct` tuple.
- `BinaryStructure` instances access fields through named properties instead of indexed tuples. `struct` tuples are fine for structures a few fields long, but when a packed binary structure grows to dozens of fields, navigating its `struct` tuple grows perilous.
- `BinaryStructure` unpacks fields only when they're accessed, allowing us to define libraries of structures scores of fields long, understanding that any particular application might access only one or two of them.
- Fields in a `BinaryStructure` can overlap eachother, greatly simplifying both C unions and fields with multiple interpretations (integer/string, signed/unsigned).
- This module makes sparse structures easy. If you're reverse-engineering a large binary structure and discover a 4-byte integer in the middle of 68 bytes of unidentified mess, this module makes it easy to add an `IntegerField` at a known structure offset. `struct` requires you to split your `'68x'` into a `'32xI32x'` (or was that a `'30xi34x'`? Better recount.)

This package exports the following names:

- `BinaryStructure` – a base class for binary data structures
- `ByteField` – a field that maps fixed-length binary data to Python strings
- `LengthPrependedStringField` – a field that maps variable-length binary strings to Python strings
- `IntegerField` – a field that maps fixed-length binary data to Python numbers

1.2.1 `BinaryStructure` Subclasses

`bodatools.binfile.eudora` – Eudora email index files

Map binary email table of contents files for the Eudora mail client to Python objects.

The Eudora email client has a long history through the early years of email. It supported versions for early Mac systems as well as early Windows OSes. Unfortunately, most of them use binary file formats that are entirely incompatible with one another. This module is aimed at one day reading all of them, but for now practicality and immediate needs demand that it focus on the files saved by a particular version on mid-90s Mac System 7.

That Eudora version stores email in flat (non-hierarchical) folders. It stores each folder's email data in a single file akin to a Unix mbox file, but with some key differences, described below. In addition to this folder data file, each folder also stores a binary "table of contents" index. In this version, a folder called `In` stores its index in a file called `In.toc`. This file consists of a fixed-size binary header with folder metadata, followed by fixed-size binary email records containing cached email header metadata as well as the location of the full email in the mbox-like data file. As the contents of the folder are updated, these fixed-size binary email records are added, removed, and reordered, apparently compacting the file as necessary so that it matches the folder contents displayed to the application end user.

With the index serving to dictate the order of the emails and their contents, their locations and sizes inside the data storage file become less important. When emails are deleted from a folder, the index is updated, but they are not removed immediately from the data file. Instead that data space is marked as inactive and might be reused later when a new email is added to the folder. As a result, the folder data file may contain stale and out-of-order data and thus **cannot be read directly as a standard mbox file**.

This module, then, provides classes for parsing the binary structures of the index file and mapping them to Python objects. This binary file has gone through many formats. Only one is represented in this module, though it could certainly be expanded to support more. Parsers and information about other versions of the index file are available at http://eudora2unix.sourceforge.net/ and http://users.starpower.net/ksimler/eudora/toc.html; these were immensely helpful in reverse-engineering the version represented by this module.

This module exports the following names:

- `Toc` – a `BinaryStructure` for the index file header
- `Message` – a `BinaryStructure` for the fixed-length email metadata entries in the index files

class `bodatools.binfile.eudora.`**`Message`** (*fobj=None*, *mm=None*, *offset=0*)

A `BinaryStructure` for a single email's metadata cached in the index file.

Only a few fields are currently represented; other fields contain interesting data but have not yet been reverse-engineered.

`LENGTH` **= 220**
the size of a single message header

`body_offset`
the offset of the body within the raw email data

`date`
a date value copied from email headers

`offset`
the offset of the raw email data in the folder data file

`priority`
some kind of unspecified single-byte priority field

`size`
the size of the raw email data in the folder data file

status
some kind of unspecified single-byte status field

subject
the email subject copied from email headers

to
a recipient copied from email headers

class `bodatools.binfile.eudora.`**Toc** (*fobj=None*, *mm=None*, *offset=0*)
A `BinaryStructure` for an email folder index header.

Only a few fields are currently represented; other fields contain interesting data but have not yet been reverse-engineered.

LENGTH = 278
the size of this binary header

messages
a generator yielding the `Message` structures in the index

name
the user-displayed folder name, e.g., "In" for the default inbox

version
the file format version

`bodatools.binfile.outlookexpress` – Outlook Express 4.5 for Mac

Map binary email folder index and content files for Outlook Express 4.5 for Macintosh to Python objects.

What documentation is available suggests that Outlook Express stored email in either .mbx or .dbx format, but in Outlook Express 4.5 for Macintosh, each mail folder consists of a directory with an `Index` file and an optional `Mail` file (no Mail file is present when a mail folder is empty).

class `bodatools.binfile.outlookexpress.`**MacFolder** (*folder_path*)
Wrapper object for an Outlook Express 4.5 for Mac folder, with a `MacIndex` and an optional `MacMail`.

Parameters **folder_path** – path to the Outlook Express 4.5 folder directory, which must contain at least an `Index` file (and probably a `Mail` file, for non-empty folders)

all_messages
Same as `messages` except deleted messages are included.

count
Number of email messages in this folder

messages
A generator yielding an `email.message.Message` for each message in this folder, based on message index information in `MacIndex` and content in `MacMail`. Does **not** include deleted messages.

raw_messages
A generator yielding a `MacMailMessage` binary object for each message in this folder, based on message index information in `MacIndex` and content in `MacMail`.

skipped_chunks = None
Number of data chunks skipped between raw messages, based on offset and size. (Only set after iterating through messages.)

class `bodatools.binfile.outlookexpress.`**MacIndex** (*fobj=None*, *mm=None*, *offset=0*)
A `BinaryStructure` for the Index file of an Outlook Express 4.5 for Mac email folder.

`MAGIC_NUMBER` = **'FMIn'**
Magic Number for Outlook Express 4.5 Mac Index file

`header_length` = **28**
length of the binary header at the beginning of the Index file

`messages`
A generator yielding the `MacIndexMessage` structures in this index file.

`total_messages`
number of email messages in this folder

class `bodatools.binfile.outlookexpress.`**`MacIndexMessage`**(*fobj=None*, *mm=None*, *offset=0*)
Information about a single email message within the `MacIndex`.

`LENGTH` = **52**
size of a single message information block

`offset`
the offset of the raw email data in the folder data file

`size`
the size of the raw email data in the folder data file

class `bodatools.binfile.outlookexpress.`**`MacMail`**(*fobj=None*, *mm=None*, *offset=0*)
A `BinaryStructure` for the Mail file of an Outlook Express 4.5 for Mac email folder. The Mail file includes the actual contents of any email files in the folder, which must be accessed based on the message offset and size from the Index file.

`MAGIC_NUMBER` = **'FMDF'**
Magic Number for a mail content file within an Outlook Express 4.5 for Macintosh folder

`get_message`(*offset*, *size*)
Get an individual `MacMailMessage` within a Mail data file, based on size and offset information from the corresponding `MacIndexMessage`.

Parameters

- **offset** – offset within the Mail file where the desired message begins, i.e. `MacMailMessage.offset`
- **size** – size of the message, i.e. `MacMailMessage.size`

class `bodatools.binfile.outlookexpress.`**`MacMailMessage`**(*size*, **args*, ***kwargs*)
A single email message within the Mail data file, as indexed by a `MacIndexMessage`. Consists of a variable length header or message summary followed by the content of the email (also variable length).

The size of a single `MacMailMessage` is stored in the `MacIndexMessage` but not (as far as we have determined) in the Mail data file, an individual message must be initialized with the a size parameter, so that the correct content can be returned.

Parameters **size** – size of this message (as determined by `MacIndexMessage.size`); **required** to return `data` correctly.

`DELETED_MESSAGE` = **'MDel'**
Header string indicating a deleted message

`MESSAGE` = **'MSum'**
Header string indicating a normal message

`as_email`()
Return message data as a `email.message.Message` object.

content_offset
offset within this message block where the message summary header ends and message content begins

data
email content for this message

deleted
boolean flag indicating if this is a deleted message

header_type
Each mail message begins with a header, starting with either `MSum` (message summary, perhaps) or `MDel` for deleted messages.

1.2.2 General Usage

Suppose we have an 8-byte file whose binary data consists of the bytes 0, 1, 2, 3, etc.:

```
>>> with open('numbers.bin') as f:
...     f.read()
...
'\x00\x01\x02\x03\x04\x05\x06\x07'
```

Suppose further that these contents represent sensible binary data, laid out such that the first two bytes are a literal string value. Except that sometimes, in the binary format we're parsing, it might sometimes be necessary to interpret those first two bytes not as a literal string, but instead as a number, encoded as a big-endian unsigned integer. Following that is a variable-length string, encoded with the total string length in the third byte.

This structure might be represented as:

```
from bodatools.binfile import *
class MyObject(BinaryStructure):
    mybytes = ByteField(0, 2)
    myint = IntegerField(0, 2)
    mystring = LengthPrepededStringField(2)
```

Client code might then read data from that file:

```
>>> f = open('numbers.bin')
>>> obj = MyObject(f)
>>> obj.mybytes
'\x00\x01'
>>> obj.myint
1
>>> obj.mystring
'\x03\x04'
```

It's not uncommon for such binary structures to be repeated at different points within a file. Consider if we overlay the same structure on the same file, but starting at byte 1 instead of byte 0:

```
>>> f = open('numbers.bin')
>>> obj = MyObject(f, offset=1)
>>> obj.mybytes
'\x01\x02'
>>> obj.myint
258
>>> obj.mystring
'\x04\x05\x06'
```

1.2.3 `BinaryStructure`

class `bodatools.binfile.`**`BinaryStructure`**(*fobj=None*, *mm=None*, *offset=0*)

A superclass for binary data structures superimposed over files.

Typical users will create a subclass containing field objects (e.g., `ByteField`, `IntegerField`). Each subclass instance is created with a file and with an optional offset into that file. When code accesses fields on the instance, they are calculated from the underlying binary file data.

Instead of a file, it is occasionally appropriate to overlay an `mmap` structure (from the `mmap` standard library). This happens most often when one `BinaryStructure` instance creates another, passing `self.mmap` to the secondary object's constructor. In this case, the caller may specify the *mm* argument instead of an *fobj*.

Parameters

- **fobj** – a file object or filename to overlay
- **mm** – a `mmap` object to overlay
- **offset** – the offset into the file where the structured data begins

1.2.4 Field classes

class `bodatools.binfile.`**`ByteField`**(*start*, *end*)

A field mapping fixed-length binary data to Python strings.

Parameters

- **start** – The offset into the structure of the beginning of the byte data.
- **end** – The offset into the structure of the end of the byte data. This is actually one past the last byte of data, so a four-byte `ByteField` starting at index 4 would be defined as `ByteField(4, 8)` and would include bytes 4, 5, 6, and 7 of the binary structure.

Typical users will create a *ByteField* inside a `BinaryStructure` subclass definition:

```
class MyObject(BinaryStructure):
    myfield = ByteField(0, 4) # the first 4 bytes of the file
```

When you instantiate the subclass and access the field, its value will be the literal bytes at that location in the structure:

```
>>> o = MyObject('file.bin')
>>> o.myfield
'ABCD'
```

class `bodatools.binfile.`**`LengthPrependedStringField`**(*offset*)

A field mapping variable-length binary strings to Python strings.

This field accesses strings encoded with their length in their first byte and string data following that byte.

Parameters **offset** – The offset of the single-byte string length.

Typical users will create a `LengthPrependedStringField` inside a `BinaryStructure` subclass definition:

```
class MyObject(BinaryStructure):
    myfield = LengthPrependedStringField(0)
```

When you instantiate the subclass and access the field, its length will be read from that location in the structure, and its data will be the bytes immediately following it. So with a file whose first bytes are `'\x04ABCD'`:

```
>>> o = MyObject('file.bin')
>>> o.myfield
'ABCD'
```

class `bodatools.binfile.`**`IntegerField`**`(`*start*, *end*`)`

A field mapping fixed-length binary data to Python numbers.

This field accessses arbitrary-length integers encoded as binary data. Currently only big-endian, unsigned integers are supported.

Parameters

- **start** – The offset into the structure of the beginning of the byte data.
- **end** – The offset into the structure of the end of the byte data. This is actually one past the last byte of data, so a four-byte `IntegerField` starting at index 4 would be defined as `IntegerField(4, 8)` and would include bytes 4, 5, 6, and 7 of the binary structure.

Typical users will create an *IntegerField* inside a `BinaryStructure` subclass definition:

```
class MyObject(BinaryStructure):
    myfield = IntegerField(3, 6) # integer encoded in bytes 3, 4, 5
```

When you instantiate the subclass and access the field, its value will be big-endian unsigned integer encoded at that location in the structure. So with a file whose bytes 3, 4, and 5 are `'\x00\x01\x04'`:

```
>>> o = MyObject('file.bin')
>>> o.myfield
260
```

1.3 Change & Version Information

The following is a summary of changes and improvements to `bodatools`. New features in each version should be listed, with any necessary information about installation or upgrade notes.

1.3.1 0.1

- Initial import of `bodatools.binfile` from `eulcommon`.

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX