Compoundfiles Documentation

Release 0.3

Dave Jones

Mar 05, 2018

Contents

This package provides a library for reading Microsoft's Compound File Binary format (CFB), formerly known as OLE Compound Documents, the Advanced Authoring Format (AAF), or just plain old Microsoft Office files (the non-XML sort). This format is also widely used with certain media systems and a number of scientific applications (tomography and microscopy).

The code is pure Python and should run on any platform; it is compatible with Python 2.7 (or above) and Python 3.2 (or above). The library has an emphasis on rigour and performs numerous validity checks on opened files. By default, the library merely warns when it comes across non-fatal errors in source files but this behaviour is configurable by developers through Python's `warnings` mechanisms.

CHAPTER 1

Links

- The code is licensed under the MIT license
- The source code can be obtained from GitHub, which also hosts the bug tracker
- The documentation (which includes installation instructions and quick-start examples) can be read on ReadTheDocs
- The build status can be observed on Travis CI

CHAPTER 2

Table of Contents

2.1 Installation

The library has no dependencies in and of itself, and consists entirely of Python code, so installation should be trivial on most platforms. A debian packaged variant is available from the author's PPA for Ubuntu users, otherwise installation from PyPI is recommended.

2.1.1 Ubuntu installation

To install from the author's PPA:

```
$ sudo add-apt-repository ppa://waveform/ppa
$ sudo apt-get update
$ sudo apt-get install python-compoundfiles
```

To remove the installation:

```
$ sudo apt-get remove python-compoundfiles
```

2.1.2 PyPI installation

To install from PyPI:

```
$ sudo pip install compoundfiles
```

To upgrade the installation:

```
$ sudo pip install -U compoundfiles
```

To remove the installation:

```
$ sudo pip uninstall compoundfiles
```

2.1.3 Development installation

If you wish to develop the library yourself, you are best off doing so within a virtualenv with source checked out from GitHub, like so:

```
$ sudo apt-get install make git python-virtualenv exuberant-ctags
$ virtualenv sandbox
$ source sandbox/bin/activate
$ git clone https://github.com/waveform-computing/compoundfiles.git
$ cd compoundfiles
$ make develop
```

The above instructions assume Ubuntu Linux, and use the included Makefile to perform a development installation into the constructed virtualenv (as well as constructing a tags file for easier navigation in vim/emacs). Please feel free to extend this section with instructions for alternate platforms.

2.2 Quick Start

Import the library and open a compound document file:

```
>>> import compoundfiles
>>> doc = compoundfiles.CompoundFileReader('foo.txm')
compoundfiles/__init__.py:606: CompoundFileWarning: DIFAT terminated by FREE_SECTOR
  CompoundFileWarning)
```

When opening the file you may see various warnings printed to the console (as in the example above). The library performs numerous checks for compliance with the specification, but many implementations don't *quite* conform. By default the warnings are simply printed and can be ignored, but via Python's `warnings` system you can either silence the warnings entirely or convert them into full blown exceptions.

You can list the contents of the compound file via the *`root`* attribute which can be treated like a dictionary:

```
>>> doc.root
["<CompoundFileEntity name='Version'>",
 u"<CompoundFileEntity dir='AutoRecon'>",
 u"<CompoundFileEntity dir='ImageInfo'>",
 u"<CompoundFileEntity dir='ImageData1'>",
 u"<CompoundFileEntity dir='ImageData2'>",
 u"<CompoundFileEntity dir='ImageData3'>",
 u"<CompoundFileEntity dir='ImageData4'>",
 u"<CompoundFileEntity dir='ImageData5'>",
 u"<CompoundFileEntity dir='ImageData6'>",
 u"<CompoundFileEntity dir='ImageData7'>",
 u"<CompoundFileEntity dir='ImageData8'>",
 u"<CompoundFileEntity dir='ImageData9'>",
 u"<CompoundFileEntity dir='SampleInfo'>",
 u"<CompoundFileEntity dir='ImageData10'>",
 u"<CompoundFileEntity dir='ImageData11'>",
 u"<CompoundFileEntity dir='ImageData12'>",
 u"<CompoundFileEntity dir='ImageData13'>",
 u"<CompoundFileEntity dir='ImageData14'>",
 u"<CompoundFileEntity dir='ImageData15'>",
 u"<CompoundFileEntity dir='ImageData16'>",
 u"<CompoundFileEntity dir='ImageData17'>",
 u"<CompoundFileEntity dir='ImageData18'>",
 u"<CompoundFileEntity dir='ImageData19'>",
 u"<CompoundFileEntity dir='ImageData20'>"]
>>> doc.root['ImageInfo']
["<CompoundFileEntity name='Date'>",
 "<CompoundFileEntity name='Angles'>",
 "<CompoundFileEntity name='Energy'>",
```

```
 "<CompoundFileEntity name='Current'>",
 "<CompoundFileEntity name='Voltage'>",
 "<CompoundFileEntity name='CameraNo'>",
 "<CompoundFileEntity name='DataType'>",
 "<CompoundFileEntity name='ExpTimes'>",
 "<CompoundFileEntity name='PixelSize'>",
 "<CompoundFileEntity name='XPosition'>",
 "<CompoundFileEntity name='YPosition'>",
 "<CompoundFileEntity name='ZPosition'>",
 "<CompoundFileEntity name='ImageWidth'>",
 "<CompoundFileEntity name='MosiacMode'>",
 "<CompoundFileEntity name='MosiacRows'>",
 "<CompoundFileEntity name='NoOfImages'>",
 "<CompoundFileEntity name='FocusTarget'>",
 "<CompoundFileEntity name='ImageHeight'>",
 "<CompoundFileEntity name='ImagesTaken'>",
 "<CompoundFileEntity name='ReadoutFreq'>",
 "<CompoundFileEntity name='ReadOutTime'>",
 "<CompoundFileEntity name='Temperature'>",
 "<CompoundFileEntity name='DtoRADistance'>",
 "<CompoundFileEntity name='HorizontalBin'>",
 "<CompoundFileEntity name='MosiacColumns'>",
 "<CompoundFileEntity name='NanoImageMode'>",
 "<CompoundFileEntity name='ObjectiveName'>",
 "<CompoundFileEntity name='ReferenceFile'>",
 "<CompoundFileEntity name='StoRADistance'>",
 "<CompoundFileEntity name='VerticalalBin'>",
 "<CompoundFileEntity name='BackgroundFile'>",
 "<CompoundFileEntity name='MosaicFastAxis'>",
 "<CompoundFileEntity name='MosaicSlowAxis'>",
 "<CompoundFileEntity name='AcquisitionMode'>",
 "<CompoundFileEntity name='TubelensPosition'>",
 "<CompoundFileEntity name='IonChamberCurrent'>",
 "<CompoundFileEntity name='NoOfImagesAveraged'>",
 "<CompoundFileEntity name='OpticalMagnification'>",
 "<CompoundFileEntity name='AbsorptionScaleFactor'>",
 "<CompoundFileEntity name='AbsorptionScaleOffset'>",
 "<CompoundFileEntity name='TransmissionScaleFactor'>",
 "<CompoundFileEntity name='OriginalDataRefCorrected'>",
 "<CompoundFileEntity name='RefTypeToApplyIfAvailable'>"]
```

Use the *open()* method with a *CompoundFileEntity*, or with a name that leads to one, to obtain a file-like object which can read the stream's content:

```
>>> doc.open('AutoRecon/BeamHardeningFilename').read()
'Standard Beam Hardening Correction\x00'
>>> f = doc.open(doc.root['ImageData1']['Image1'])
>>> f.tell()
0
>>> import os
>>> f.seek(0, os.SEEK_END)
8103456
>>> f.seek(0)
0
>>> f.read(10)
'\xb3\x0c\xb3\x0c\xb3\x0c\xb3\x0c\xb3\x0c'
>>> f.close()
```

You can also use entities as iterators, and the context manager protocol is supported for file and stream opening:

```
>>> with compoundfiles.CompoundFileReader('foo.txm') as doc:
...     for entry in doc.root['AutoRecon']:
```

```
...         if entry.isfile:
...             with doc.open(entry) as stream:
...                 print(repr(stream.read()))
...
'"\x00>C'
'\x81\x02SG'
'\x1830\xc5'
'\x00\x00\x00\x00'
'\x9a\x99\x99?'
'\xcf.AD'
'(\x1c\x1cF'
',E\xd6\xc3'
'\x02\x00\x00\x00'
'\x01\x00\x00\x00'
'\x00\x00\x00\x00'
'\x00\x00\x00\x00'
'\xd4\xfe\x9fA'
'\xd1\x07\x00\x00'
'\x05\x00\x00\x00'
'\x00\x00\x00\x00'
'p\xff\x1fB'
'\x00\x00\x00\x00'
'\x02\x00\x00\x00'
'\x01\x00\x00\x00'
'Standard Beam Hardening Correction\x00'
'\x00'
```

2.3 Compliance mechanisms

As noted in the CFB specification, the compound document format presents a number of validation challenges. For example, maliciously constructed files might include circular references in their FAT table, leading a naive reader into an infinite loop, or they may allocate a large number of DIFAT sectors hoping to cause resource exhaustion when the reader goes to allocate memory for reading the FAT.

The compoundfiles library goes to some lengths to detect erroneous structures (whether malicious in intent or otherwise) and work around them where possible. Some issues are considered fatal and will always raise an exception (circular chains in the FAT are an example of this). Other issues are considered non-fatal and will raise a warning (unusual sector sizes are an example of this). Python `warnings` are a special sort of exception with particularly flexible handling.

With Python's defaults, a specific warning will print a message to the console the first time it is encountered and will then do nothing if it's encountered again (this avoids spamming the console in case a warning is raised in a tight loop). With some simple code, you can specify alternative behaviours: warnings can be raised as full-blown exceptions, or suppressed entirely. The compoundfiles library defines a large hierarchy of errors and warnings to enable developers to finetune their handling.

For example, consider a developer writing an application for working with computed tomography (CT) scans. The files produced by the scanner's software are compound documents, but they use an unusual sector size. Whenever the developer's Python script opens a file the following warning is emitted:

```
/usr/lib/pyshared/python2.7/compoundfiles/compoundfiles/reader.py:275:␣
↪CompoundFileSectorSizeWarning: unexpected sector size in v3 file (1024)
```

Other than this, the script runs successfully. The developer decides the warning is unimportant (after all there's nothing he can do about it given he can't change the scanner's software) and wishes to suppress it entirely, so he adds the following line to the top of his script:

```
import warnings
import compoundfiles as cf
```

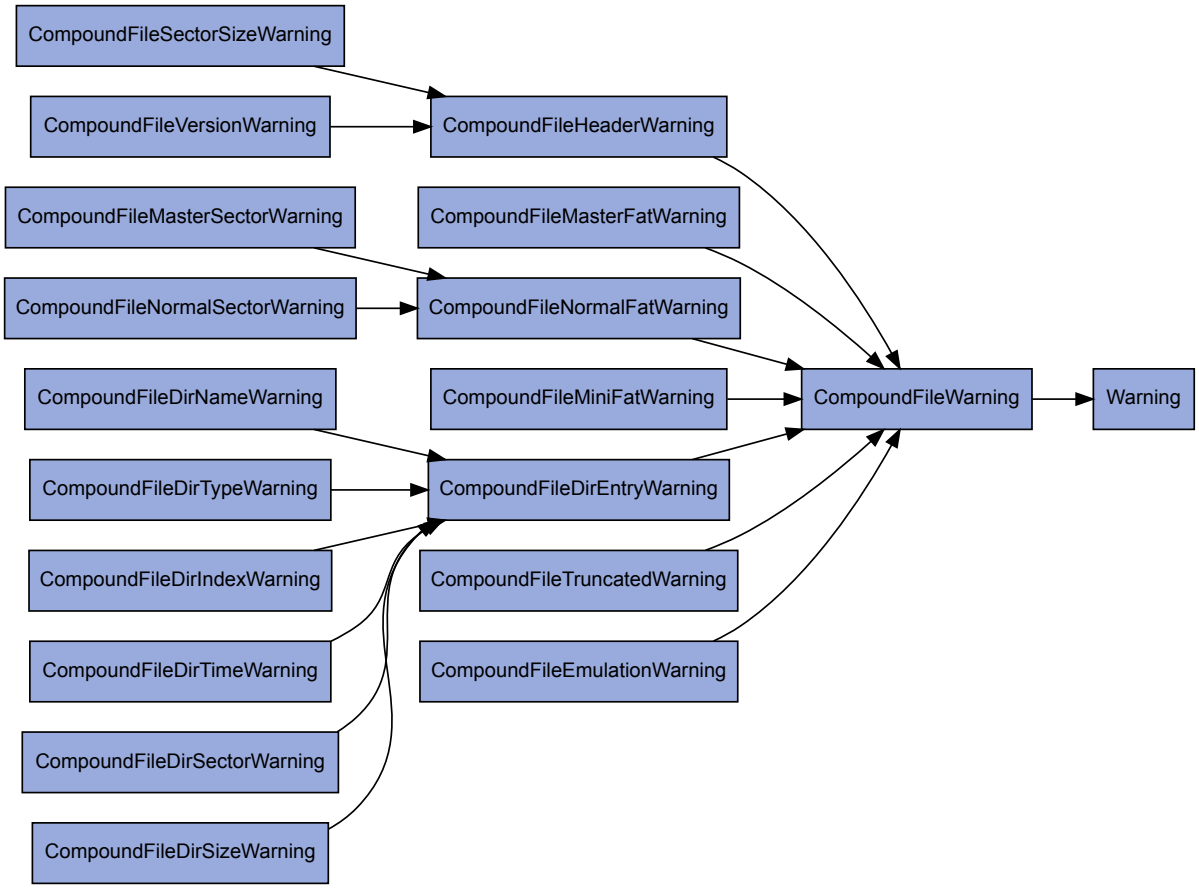
```
warnings.filterwarnings('ignore', category=cf.CompoundFileSectorSizeWarning)
```

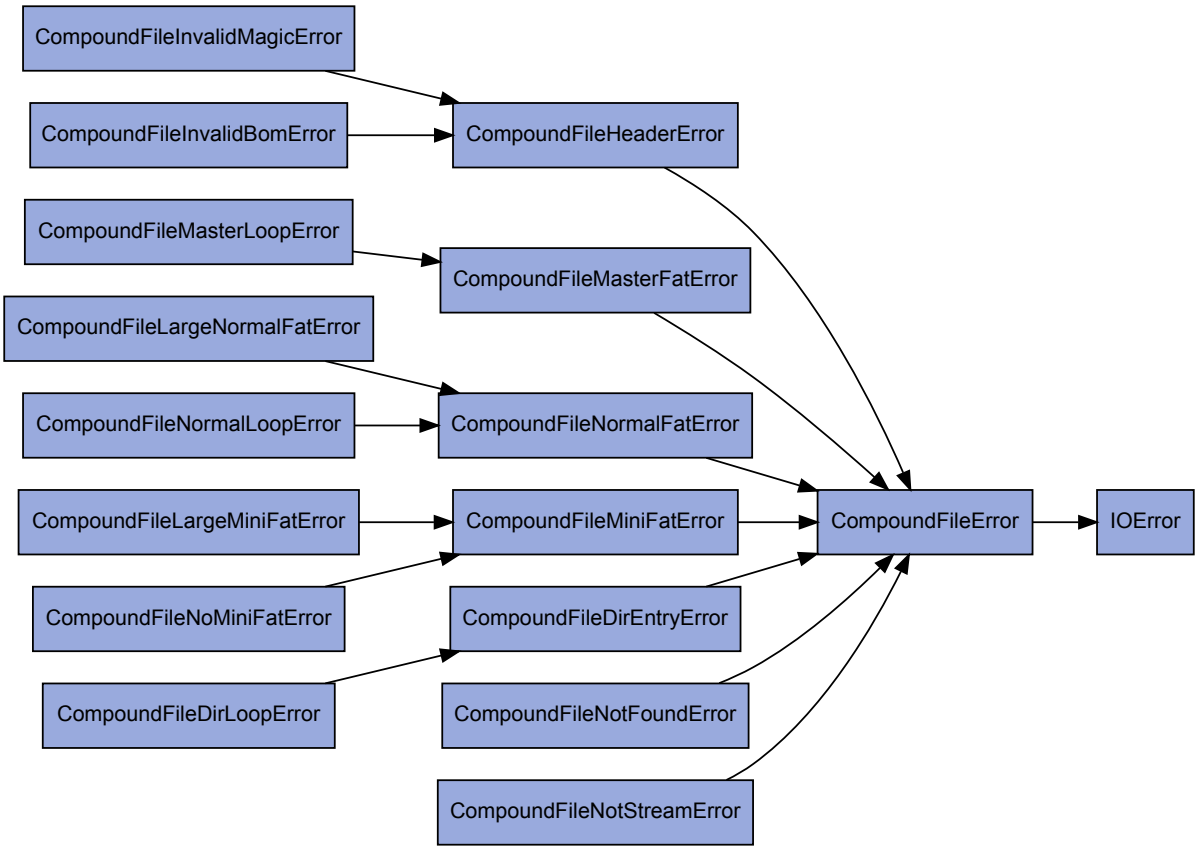
Another developer is working on a file validation service. She wishes to use the compoundfiles library to extract and examine the contents of such files. For safety, she decides to treat any violation of the specification as an error, so she adds the following line to the top of her script to tell Python to convert all compound file warnings into exceptions:

```
import warnings
import compoundfiles as cf

warnings.filterwarnings('error', category=cf.CompoundFileWarning)
```

The class hierarchies for compoundfiles warnings and errors is illustrated below:





To set filters on all warnings in the hierarchy, simply use the category *CompoundFileWarning*. Otherwise, you can use intermediate or leaf classes in the hierarchy for more specific filters. Likewise, when catching exceptions you can target the root of the hierarchy (*CompoundFileError*) to catch any error that the compoundfiles library might raise, or a more specific class to deal with a particular error.

2.4 API Reference

Most of the work in this package was derived from the specification for OLE Compound Document files published by OpenOffice, and the specification for the Advanced Authoring Format (AAF) published by Microsoft.

2.4.1 CompoundFileReader

class compoundfiles.**CompoundFileReader**(*filename_or_obj*)

Provides an interface for reading OLE Compound Document files.

The *CompoundFileReader* class provides a relatively simple interface for interpreting the content of Microsoft's OLE Compound Document files. These files can be thought of as a file-system in a file (or a loop-mounted FAT file-system for Unix folk).

The class can be constructed with a filename or a file-like object. In the latter case, the object must support the `read`, `seek`, and `tell` methods. For optimal usage, it should also provide a valid file descriptor in response to a call to `fileno`, but this is not mandatory.

The *root* attribute represents the root storage entity in the compound document. An *open()* method is provided which (given a *CompoundFileEntity* instance representing a stream), returns a file-like object representing the content of the stream.

Finally, the context manager protocol is also supported, permitting usage of the class like so:

```
with CompoundFileReader('foo.doc') as doc:
    # Iterate over items in the root directory of the compound document
    for entry in doc.root:
        # If any entry is a file, attempt to read the data from it
        if entry.isfile:
            with doc.open(entry) as f:
                f.read()
```

root
The root attribute represents the root storage entity in the compound document. As a *CompoundFileEntity* instance, it (and child storages) can be enumerated, accessed by index, or by name (like a dict) to obtain *CompoundFileEntity* instances representing the content of the compound document.

Both *CompoundFileReader* and *CompoundFileEntity* support human-readable representations making it relatively simple to browse and extract information from compound documents simply by using the interactive Python command line.

open(*filename_or_entity*)
Return a file-like object with the content of the specified entity.

Given a *CompoundFileEntity* instance which represents a stream, or a string representing the path to one (using / separators), this method returns an instance of *CompoundFileStream* which can be used to read the content of the stream.

2.4.2 CompoundFileStream

class compoundfiles.**CompoundFileStream**
Abstract base class for streams within an OLE Compound Document.

Instances of *CompoundFileStream* are not constructed directly, but are returned by the *CompoundFileReader.open()* method. They support all common methods associated with read-only streams (*read()*, *seek()*, *tell()*, and so forth).

read(*n=-1*)
Read up to *n* bytes from the stream and return them. As a convenience, if *n* is unspecified or -1, `readall()` is called. Fewer than *n* bytes may be returned if there are fewer than *n* bytes from the current stream position to the end of the stream.

If 0 bytes are returned, and *n* was not 0, this indicates end of the stream.

read1(*n=-1*)
Read up to *n* bytes from the stream using only a single call to the underlying object.

In the case of *CompoundFileStream* this roughly corresponds to returning the content from the current position up to the end of the current sector.

readable()
Returns `True`, indicating that the stream supports *read()*.

seek(*offset*, *whence=0*)
Change the stream position to the given byte *offset*. *offset* is interpreted relative to the position indicated by *whence*. Values for *whence* are:

- `SEEK_SET` or `0` - start of the stream (the default); *offset* should be zero or positive
- `SEEK_CUR` or `1` - current stream position; *offset* may be negative
- `SEEK_END` or `2` - end of the stream; *offset* is usually negative

Return the new absolute position.

seekable()
Returns `True`, indicating that the stream supports *seek()*.

tell()
Return the current stream position.

writable()
Returns `False`, indicating that the stream doesn't support `write()` or `truncate()`.

2.4.3 CompoundFileEntity

class compoundfiles.**CompoundFileEntity**(*parent*, *stream*, *index*)
Represents an entity in an OLE Compound Document.

An entity in an OLE Compound Document can be a "stream" (analogous to a file in a file-system) which has a *size* and can be opened by a call to the parent object's *open()* method. Alternatively, it can be a "storage" (analogous to a directory in a file-system), which has no size but has *created* and *modified* time-stamps, and can contain other streams and storages.

If the entity is a storage, it will act as an iterable read-only sequence, indexable by ordinal or by name, and compatible with the `in` operator and built-in `len()` function.

created
For storage entities (where *isdir* is `True`), this returns the creation date of the storage. Returns `None` for stream entities.

isdir
Returns True if this is a storage entity which can contain other entities.

isfile
Returns True if this is a stream entity which can be opened.

modified
For storage entities (where *isdir* is True), this returns the last modification date of the storage. Returns `None` for stream entities.

name
Returns the name of entity. This can be up to 31 characters long and may contain any character representable in UTF-16 except the NULL character. Names are considered case-insensitive for comparison purposes.

size
For stream entities (where *isfile* is `True`), this returns the number of bytes occupied by the stream. Returns 0 for storage entities.

2.4.4 Exceptions

exception compoundfiles.**CompoundFileError**
Base class for exceptions arising from reading compound documents.

exception compoundfiles.**CompoundFileWarning**
Base class for warnings arising from reading compound documents.

2.5 Change log

2.5.1 Release 0.3 (2014-09-01)

- Added a comprehensive test suite and fixed several small bugs as a result (all to do with invalid file handling) (#2)
- Added an mmap emulation to enable reading of massive files on 32-bit systems; the emulation is necessarily slower than "proper" mmap but that's the cost of staying on 32-bit! (#6)

- Extended the warning and error hierarchy so that users of the library can fine tune exactly what warnings they want to consider errors (#3)

2.5.2 Release 0.2 (2014-04-23)

- Fixed a nasty bug where opening multiple streams in a compound document would result in shared file pointer state (#4)
- Fixed Python 3 compatibility - many thanks to Martin Panter for the bug report! (#5)

2.5.3 Release 0.1 (2014-02-22)

Initial release.

2.6 License

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index