olefile Documentation

Release 0.46

Philippe Lagadec

Sep 17, 2018

Contents:

1 Features 3

2 History 5

3 License for olefile 7

4 How to Download and Install olefile 9

5 How to Suggest Improvements, Report Issues or Contribute 11

6 How to use olefile - API overview 13

7 About the structure of OLE files 21

8 olefile API Reference 23

9 Frequently Asked Questions 31

Python Module Index 33

This is the home page of the documentation for olefile. The latest version can be found online, otherwise a copy is provided in the doc subfolder of the package.

olefile is a Python package to parse, read and write Microsoft OLE2 files (also called Structured Storage, Compound File Binary Format or Compound Document File Format), such as Microsoft Office 97-2003 documents, Image Composer and FlashPix files, Outlook messages, StickyNotes, several Microscopy file formats, McAfee antivirus quarantine files, etc.

Quick links: Home page - Download/Install - Documentation - Report Issues/Suggestions/Questions - Contact the author - Repository - Updates on Twitter

CHAPTER 1

Features

- Parse, read and write any OLE file such as Microsoft Office 97-2003 legacy document formats (Word .doc, Excel .xls, PowerPoint .ppt, Visio .vsd, Project .mpp), Image Composer and FlashPix files, Outlook messages, StickyNotes, Zeiss AxioVision ZVI files, Olympus FluoView OIB files, etc
- List all the streams and storages contained in an OLE file
- Open streams as files
- Parse and read property streams, containing metadata of the file
- Portable, pure Python module, no dependency

olefile can be used as an independent module or with PIL / Pillow.

olefile is mostly meant for developers. If you are looking for tools to analyze OLE files or to extract data (especially for security purposes such as malware analysis and forensics), then please also check my python-oletools, which are built upon olefile and provide a higher-level interface.

CHAPTER 2

History

olefile is based on the OleFileIO module from PIL v1.1.7, the excellent Python Imaging Library, created and maintained by Fredrik Lundh. The olefile API is still compatible with PIL, but since 2005 I have improved the internal implementation significantly, with new features, bugfixes and a more robust design.

From 2005 to 2014 the project was called **OleFileIO_PL**, and in 2014 I changed its name to **olefile** to celebrate its 9 years and its new write features.

As far as I know, this module is the most complete and robust Python implementation to read MS OLE2 files, portable on several operating systems. (please tell me if you know other similar Python modules)

Since 2014 olefile/OleFileIO_PL has been integrated into Pillow, the friendly fork of PIL.

In January 2017, it was decided to remove olefile from Pillow 4.0.0 and to install it as an external dependency. This will avoid issues due to the maintenance of the olefile code in two repositories.

2.1 Main improvements over the original version of OleFileIO in PIL:

- Compatible with Python 3.4+ and 2.7+
- Many bug fixes
- Support for files larger than 6.8MB
- Support for 64 bits platforms and big-endian CPUs
- Robust: many checks to detect malformed files
- Runtime option to choose if malformed files should be parsed or raise exceptions
- Improved API
- Metadata extraction, stream/storage timestamps (e.g. for document forensics)
- Can open file-like objects
- Added setup.py and install.bat to ease installation

- More convenient slash-based syntax for stream paths
- Write features

2.2 Detailed History

- **2018-09-09 v0.46**: OleFileIO can now be used as a context manager (with. . . as), to close the file automatically (see doc). Improved handling of malformed files, fixed several bugs.
- 2018-01-24 v0.45: olefile can now overwrite streams of any size, improved handling of malformed files, fixed several bugs, end of support for Python 2.6 and 3.3.
- 2017-01-06 v0.44: several bugfixes, removed support for Python 2.5 (olefile2), added support for incomplete streams and incorrect directory entries (to read malformed documents), added getclsid, improved documentation with API reference.
- 2017-01-04: moved the documentation to ReadTheDocs
- 2016-05-20: moved olefile repository to GitHub
- 2016-02-02 v0.43: fixed issues #26 and #27, better handling of malformed files, use python logging.
- 2015-01-25 v0.42: improved handling of special characters in stream/storage names on Python 2.x (using UTF-8 instead of Latin-1), fixed bug in listdir with empty storages.
- 2014-11-25 v0.41: OleFileIO.open and isOleFile now support OLE files stored in byte strings, fixed installer for python 3, added support for Jython (Niko Ehrenfeuchter)
- 2014-10-01 v0.40: renamed OleFileIO_PL to olefile, added initial write support for streams >4K, updated doc and license, improved the setup script.
- 2014-07-27 v0.31: fixed support for large files with 4K sectors, thanks to Niko Ehrenfeuchter, Martijn Berger and Dave Jones. Added test scripts from Pillow (by hugovk). Fixed setup for Python 3 (Martin Panter)
- 2014-02-04 v0.30: now compatible with Python 3.x, thanks to Martin Panter who did most of the hard work.
- 2013-07-24 v0.26: added methods to parse stream/storage timestamps, improved listdir to include storages, fixed parsing of direntry timestamps
- 2013-05-27 v0.25: improved metadata extraction, properties parsing and exception handling, fixed issue #12
- 2013-05-07 v0.24: new features to extract metadata (get_metadata method and OleMetadata class), improved getproperties to convert timestamps to Python datetime
- 2012-10-09: published python-oletools, a package of analysis tools based on OleFileIO_PL
- 2012-09-11 v0.23: added support for file-like objects, fixed issue #8
- 2012-02-17 v0.22: fixed issues #7 (bug in getproperties) and #2 (added close method)
- 2011-10-20: code hosted on bitbucket to ease contributions and bug tracking
- 2010-01-24 v0.21: fixed support for big-endian CPUs, such as PowerPC Macs.
- 2009-12-11 v0.20: small bugfix in OleFileIO.open when filename is not plain str.
- 2009-12-10 v0.19: fixed support for 64 bits platforms (thanks to Ben G. and Martijn for reporting the bug)

CHAPTER 3

License for olefile

olefile (formerly OleFileIO_PL) is copyright (c) 2005-2018 Philippe Lagadec (http://www.decalage.info)

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

---

olefile is based on source code from the OleFileIO module of the Python Imaging Library (PIL) published by Fredrik Lundh under the following license:

The Python Imaging Library (PIL) is

- Copyright (c) 1997-2009 by Secret Labs AB
- Copyright (c) 1995-2009 by Fredrik Lundh

By obtaining, using, and/or copying this software and/or its associated documentation, you agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and its associated documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appears in all copies, and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Secret Labs AB or the author not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

SECRET LABS AB AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL SECRET LABS AB OR THE AUTHOR BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

CHAPTER 4

How to Download and Install olefile

4.1 Pre-requisites

olefile requires Python 2.7 or 3.4+.

4.2 Download and Install

To use olefile with other Python applications or your own scripts, the simplest solution is to run `pip install olefile` or `easy_install olefile`, to download and install the package in one go. Pip is part of the standard Python distribution since v2.7.9.

To update olefile if a previous version is already installed, run `pip install -U olefile`.

Otherwise you may download/extract the zip archive in a temporary directory and run `python setup.py install`.

On Windows you may simply double-click on `install.bat`.

CHAPTER 5

How to Suggest Improvements, Report Issues or Contribute

This is a personal open-source project, developed on my spare time. Any contribution, suggestion, feedback or bug report is welcome.

To **suggest improvements, report a bug or any issue**, please use the issue reporting page, providing all the information and files to reproduce the problem.

If possible please join the debugging output of olefile. For this, launch the following command :

```
olefile.py -d -c file >debug.txt
```

You may also contact the author directly to **provide feedback**.

The code is available in a repository on GitHub. You may use it to **submit enhancements** using forks and pull requests.

CHAPTER 6

How to use olefile - API overview

This page is part of the documentation for olefile. It explains how to use all its features to parse and write OLE files. For more information about OLE files, see *About the structure of OLE files*.

olefile can be used as an independent module or with PIL/Pillow. The main functions and methods are explained below.

For more information, see also the *olefile API Reference*, sample code at the end of the module itself, and docstrings within the code.

6.1 Import olefile

When the `olefile` package has been installed, it can be imported in Python applications with this statement:

```
import olefile
```

As of version 0.30, the code has been changed to be compatible with Python 3.x. As a consequence, compatibility with Python 2.5 or older is not provided anymore.

6.2 Test if a file is an OLE container

Use `olefile.isOleFile()` to check if the first bytes of the file contain the Magic for OLE files, before opening it. isOleFile returns True if it is an OLE file, False otherwise (new in v0.16).

```
assert olefile.isOleFile('myfile.doc')
```

The argument of isOleFile can be (new in v0.41):

- the path of the file to open on disk (bytes or unicode string smaller than 1536 bytes),
- or a bytes string containing the file in memory. (bytes string longer than 1535 bytes),
- or a file-like object (with read and seek methods).

6.3 Open an OLE file from disk

Create an *olefile.OleFileIO* object with the file path as parameter:

```
ole = olefile.OleFileIO('myfile.doc')
```

Since olefile v0.46, the recommended way to open an OLE file is to use OleFileIO as a context manager, using the "with" clause:

```
with olefile.OleFileIO('myfile.doc') as ole
    # perform all operations on the ole object
```

This guarantees that the OleFileIO object is closed when exiting the with block, even if an exception is triggered. It will call *olefile.OleFileIO.close()* automatically.

(new in v0.46)

6.4 Open an OLE file from a bytes string

This is useful if the file is already stored in memory as a bytes string.

```
ole = olefile.OleFileIO(s)
```

Note: olefile checks the size of the string provided as argument to determine if it is a file path or the content of an OLE file. An OLE file cannot be smaller than 1536 bytes. If the string is larger than 1535 bytes, then it is expected to contain an OLE file, otherwise it is expected to be a file path.

(new in v0.41)

6.5 Open an OLE file from a file-like object

This is useful if the file is not on disk but only available as a file-like object (with read, seek and tell methods).

```
ole = olefile.OleFileIO(f)
```

If the file-like object does not have seek or tell methods, the easiest solution is to read the file entirely in a bytes string before parsing:

```
data = f.read()
ole = olefile.OleFileIO(data)
```

6.6 How to handle malformed OLE files

By default, the parser is configured to be as robust and permissive as possible, allowing to parse most malformed OLE files. Only fatal errors will raise an exception. It is possible to tell the parser to be more strict in order to raise exceptions for files that do not fully conform to the OLE specifications, using the `raise_defect` option (new in v0.14):

```
ole = olefile.OleFileIO('myfile.doc', raise_defects=olefile.DEFECT_INCORRECT)
```

When the parsing is done, the list of non-fatal issues detected is available as a list in the *olefile.OleFileIO.parsing_issues* attribute of the OleFileIO object (new in 0.25):

```
print('Non-fatal issues raised during parsing:')
if ole.parsing_issues:
    for exctype, msg in ole.parsing_issues:
        print('- %s: %s' % (exctype.__name__, msg))
else:
    print('None')
```

6.7 Open an OLE file in write mode

Before using the write features, the OLE file must be opened in read/write mode, by using the option `write_mode=True`:

```
ole = olefile.OleFileIO('test.doc', write_mode=True)
```

(new in v0.40)

The code for write features is new and it has not been thoroughly tested yet. See issue #6 for the roadmap and the implementation status. If you encounter any issue, please send me your feedback or report issues.

6.8 Syntax for stream and storage paths

Two different syntaxes are allowed for methods that need or return the path of streams and storages:

1. Either a **list of strings** including all the storages from the root up to the stream/storage name. For example a stream called "WordDocument" at the root will have `['WordDocument']` as full path. A stream called "ThisDocument" located in the storage "Macros/VBA" will be `['Macros', 'VBA', 'ThisDocument']`. This is the original syntax from PIL. While hard to read and not very convenient, this syntax works in all cases.
2. Or a **single string with slashes** to separate storage and stream names (similar to the Unix path syntax). The previous examples would be `'WordDocument'` and `'Macros/VBA/ThisDocument'`. This syntax is easier, but may fail if a stream or storage name contains a slash (which is normally not allowed, according to the Microsoft specifications [MS-CFB]). (new in v0.15)

Both are case-insensitive.

Switching between the two is easy:

```
slash_path = '/'.join(list_path)
list_path  = slash_path.split('/')
```

Encoding:

- Stream and Storage names are stored in Unicode format in OLE files, which means they may contain special characters (e.g. Greek, Cyrillic, Japanese, etc) that applications must support to avoid exceptions.
- **On Python 2.x**, all stream and storage paths are handled by olefile in bytes strings, using the **UTF-8 encoding** by default. If you need to use Unicode instead, add the option `path_encoding=None` when creating the OleFileIO object. This is new in v0.42. Olefile was using the Latin-1 encoding until v0.41, therefore special characters were not supported.
- **On Python 3.x**, all stream and storage paths are handled by olefile in unicode strings, without encoding.

6.9 Get the list of streams

olefile.OleFileIO.listdir() returns a list of all the streams contained in the OLE file, including those stored in storages. Each stream is listed itself as a list, as described above.

```
print(ole.listdir())
```

Sample result:

```
[['\x01CompObj'], ['\x05DocumentSummaryInformation'], ['\x05SummaryInformation']
, ['1Table'], ['Macros', 'PROJECT'], ['Macros', 'PROJECTwm'], ['Macros', 'VBA',
'Module1'], ['Macros', 'VBA', 'ThisDocument'], ['Macros', 'VBA', '_VBA_PROJECT']
, ['Macros', 'VBA', 'dir'], ['ObjectPool'], ['WordDocument']]
```

As an option it is possible to choose if storages should also be listed, with or without streams (new in v0.26):

```
ole.listdir (streams=False, storages=True)
```

6.10 Test if known streams/storages exist:

olefile.OleFileIO.exists() checks if a given stream or storage exists in the OLE file (new in v0.16). The provided path is case-insensitive.

```
if ole.exists('worddocument'):
    print("This is a Word document.")
    if ole.exists('macros/vba'):
         print("This document seems to contain VBA macros.")
```

6.11 Read data from a stream

olefile.OleFileIO.openstream() opens a stream as a file-like object. The provided path is case-insensitive.

The following example extracts the "Pictures" stream from a PPT file:

```
pics = ole.openstream('Pictures')
data = pics.read()
```

6.12 Get information about a stream/storage

Several methods can provide the size, type and timestamps of a given stream/storage:

olefile.OleFileIO.get_size() returns the size of a stream in bytes (new in v0.16):

```
s = ole.get_size('WordDocument')
```

olefile.OleFileIO.get_type() returns the type of a stream/storage, as one of the following constants: *olefile.STGTY_STREAM* for a stream, *olefile.STGTY_STORAGE* for a storage, *olefile.STGTY_ROOT* for the root entry, and `False` for a non existing path (new in v0.15).

```
t = ole.get_type('WordDocument')
```

olefile.OleFileIO.getctime() and *olefile.OleFileIO.getmtime()* return the creation and modification timestamps of a stream/storage, as a Python datetime object with UTC timezone. Please note that these timestamps are only present if the application that created the OLE file explicitly stored them, which is rarely the case. When not present, these methods return None (new in v0.26).

```
c = ole.getctime('WordDocument')
m = ole.getmtime('WordDocument')
```

The root storage is a special case: You can get its creation and modification timestamps using the OleFileIO.root attribute (new in v0.26):

```
c = ole.root.getctime()
m = ole.root.getmtime()
```

Note: all these methods are case-insensitive.

6.13 Overwriting a sector

The *olefile.OleFileIO.write_sect()* method can overwrite any sector of the file. If the provided data is smaller than the sector size (normally 512 bytes, sometimes 4KB), data is padded with null characters. (new in v0.40)

Here is an example:

```
ole.write_sect(0x17, b'TEST')
```

Note: following the MS-CFB specifications, sector 0 is actually the second sector of the file. You may use -1 as index to write the first sector.

6.14 Overwriting a stream

The *olefile.OleFileIO.write_stream()* method can overwrite an existing stream in the file. The new stream data must be the exact same size as the existing one. Since v0.45, this method can write streams of any size (stored in the main FAT or the MiniFAT).

For example, you may change text in a MS Word document:

```
ole = olefile.OleFileIO('test.doc', write_mode=True)
data = ole.openstream('WordDocument').read()
data = data.replace(b'foo', b'bar')
ole.write_stream('WordDocument', data)
ole.close()
```

(new in v0.40)

6.15 Extract metadata

olefile.OleFileIO.get_metadata() will check if standard property streams exist, parse all the properties they contain, and return an `olefile.OleFileIO.OleMetadata` object with the found properties as attributes (new in v0.24).

```
meta = ole.get_metadata()
print('Author:', meta.author)
print('Title:', meta.title)
print('Creation date:', meta.create_time)
# print all metadata:
meta.dump()
```

Available attributes include:

```
codepage, title, subject, author, keywords, comments, template,
last_saved_by, revision_number, total_edit_time, last_printed, create_time,
last_saved_time, num_pages, num_words, num_chars, thumbnail,
creating_application, security, codepage_doc, category, presentation_target,
bytes, lines, paragraphs, slides, notes, hidden_slides, mm_clips,
scale_crop, heading_pairs, titles_of_parts, manager, company, links_dirty,
chars_with_spaces, unused, shared_doc, link_base, hlinks, hlinks_changed,
version, dig_sig, content_type, content_status, language, doc_version
```

See the source code of the `olefile.OleFileIO.OleMetadata` class for more information.

6.16 Parse a property stream

olefile.OleFileIO.getproperties() can be used to parse any property stream that is not handled by get_metadata. It returns a dictionary indexed by integers. Each integer is the index of the property, pointing to its value. For example in the standard property stream `'\x05SummaryInformation'`, the document title is property #2, and the subject is #3.

```
p = ole.getproperties('specialprops')
```

By default as in the original PIL version, timestamp properties are converted into a number of seconds since Jan 1,1601. With the option `convert_time`, you can obtain more convenient Python datetime objects (UTC timezone). If some time properties should not be converted (such as total editing time in `'\x05SummaryInformation'`), the list of indexes can be passed as no_conversion (new in v0.25):

```
p = ole.getproperties('specialprops', convert_time=True, no_conversion=[10])
```

6.17 Close the OLE file

Unless your application is a simple script that terminates after processing an OLE file, do not forget to close each OleFileIO object after parsing to close the file on disk. (new in v0.22)

```
ole.close()
```

6.18 Enable logging

See *olefile.enable_logging()*

6.19 Use olefile as a script for testing/debugging

olefile can also be used as a script from the command-line to display the structure of an OLE file and its metadata, for example:

```
olefile.py myfile.doc
```

You can use the option `-c` to check that all streams can be read fully, and `-d` to generate very verbose debugging information.

You may also add the option `-l debug` to display debugging messages (very verbose).

CHAPTER 7

About the structure of OLE files

This page is part of the documentation for olefile. It provides a brief overview of the structure of Microsoft OLE2 files (also called Structured Storage, Compound File Binary Format or Compound Document File Format), such as Microsoft Office 97-2003 documents, Image Composer and FlashPix files, Outlook messages, StickyNotes, several Microscopy file formats, McAfee antivirus quarantine files, etc.

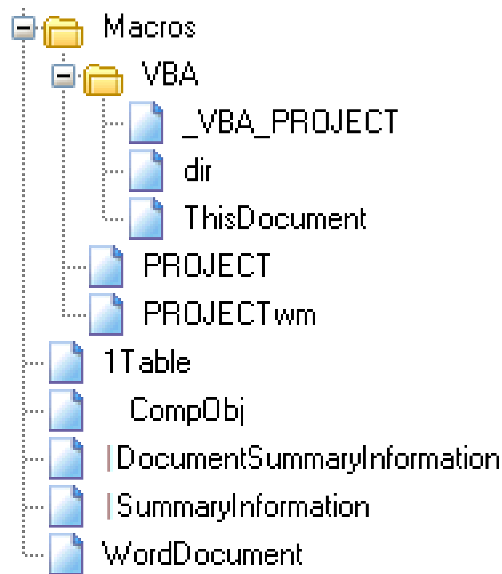
An OLE file can be seen as a mini file system or a Zip archive: It contains **streams** of data that look like files embedded within the OLE file. Each stream has a name. For example, the main stream of a MS Word document containing its text is named "WordDocument".

An OLE file can also contain **storages**. A storage is a folder that contains streams or other storages. For example, a MS Word document with VBA macros has a storage called "Macros".

Special streams can contain **properties**. A property is a specific value that can be used to store information such as the metadata of a document (title, author, creation date, etc). Property stream names usually start with the character '\x05' (ASCII code 5).

For example, a typical MS Word document may look like this:

Go to the *How to use olefile - API overview* page to see how to use all olefile features to parse OLE files.


Macros
VBA
_VBA_PROJECT
dir
ThisDocument
PROJECT
PROJECTwm
1Table
CompObj
DocumentSummaryInformation
SummaryInformation
WordDocument

CHAPTER 8

olefile API Reference

8.1 Indices and tables

- genindex
- modindex
- search

8.2 Summary

olefile.isOleFile(filename)	Test if a file is an OLE container (according to the magic bytes in its header).
olefile.OleFileIO([filename, raise_defects, …])	OLE container object
olefile.OleMetadata()	class to parse and store metadata from standard properties of OLE files.
olefile.enable_logging()	Enable logging for this module (disabled by default).

8.3 olefile module

olefile (formerly OleFileIO_PL)

Module to read/write Microsoft OLE2 files (also called Structured Storage or Microsoft Compound Document File Format), such as Microsoft Office 97-2003 documents, Image Composer and FlashPix files, Outlook messages, … This version is compatible with Python 2.7 and 3.4+

Project website: https://www.decalage.info/olefile

olefile is copyright (c) 2005-2018 Philippe Lagadec (https://www.decalage.info)

olefile is based on the OleFileIO module from the PIL library v1.1.7 See: http://www.pythonware.com/products/pil/index.htm and http://svn.effbot.org/public/tags/pil-1.1.7/PIL/OleFileIO.py

The Python Imaging Library (PIL) is Copyright (c) 1997-2009 by Secret Labs AB Copyright (c) 1995-2009 by Fredrik Lundh

See source code and LICENSE.txt for information on usage and redistribution.

`olefile.`**`isOleFile`**(*filename*)

Test if a file is an OLE container (according to the magic bytes in its header).

Note: This function only checks the first 8 bytes of the file, not the rest of the OLE structure.

New in version 0.16.

Parameters **`filename`** (*`bytes or str or unicode or file`*) – filename, contents or file-like object of the OLE file (string-like or file-like object)

- if filename is a string smaller than 1536 bytes, it is the path of the file to open. (bytes or unicode string)
- if filename is a string longer than 1535 bytes, it is parsed as the content of an OLE file in memory. (bytes type only)
- if filename is a file-like object (with read and seek methods), it is parsed as-is.

Returns True if OLE, False otherwise.

Return type bool

`class` `olefile.`**`OleFileIO`**(*filename=None*, *raise_defects=40*, *write_mode=False*, *debug=False*, *path_encoding='utf-8'*)

OLE container object

This class encapsulates the interface to an OLE 2 structured storage file. Use the listdir and openstream methods to access the contents of this file.

Object names are given as a list of strings, one for each subentry level. The root entry should be omitted. For example, the following code extracts all image streams from a Microsoft Image Composer file:

```
ole = OleFileIO("fan.mic")

for entry in ole.listdir():
    if entry[1:2] == "Image":
        fin = ole.openstream(entry)
        fout = open(entry[0:1], "wb")
        while True:
            s = fin.read(8192)
            if not s:
                break
            fout.write(s)
```

You can use the viewer application provided with the Python Imaging Library to view the resulting files (which happens to be standard TIFF files).

Constructor for the OleFileIO class.

Parameters

- **`filename`** – file to open.

  - if filename is a string smaller than 1536 bytes, it is the path of the file to open. (bytes or unicode string)
  - if filename is a string longer than 1535 bytes, it is parsed as the content of an OLE file in memory. (bytes type only)
  - if filename is a file-like object (with read, seek and tell methods), it is parsed as-is.
- **raise_defects** – minimal level for defects to be raised as exceptions. (use DEFECT_FATAL for a typical application, DEFECT_INCORRECT for a security-oriented application, see source code for details)
- **write_mode** – bool, if True the file is opened in read/write mode instead of read-only by default.
- **debug** – bool, set debug mode (deprecated, not used anymore)
- **path_encoding** – None or str, name of the codec to use for path names (streams and storages), or None for Unicode. Unicode by default on Python 3+, UTF-8 on Python 2.x. (new in olefile 0.42, was hardcoded to Latin-1 until olefile v0.41)

close()
close the OLE file, to release the file object

dumpdirectory()
Dump directory (for debugging only)

dumpfat(*fat*, *firstindex=0*)
Display a part of FAT in human-readable form for debugging purposes

dumpsect(*sector*, *firstindex=0*)
Display a sector in a human-readable form, for debugging purposes

exists(*filename*)
Test if given filename exists as a stream or a storage in the OLE container. Note: filename is case-insensitive.

Parameters **filename** – path of stream in storage tree. (see openstream for syntax)

Returns True if object exist, else False.

get_metadata()
Parse standard properties streams, return an OleMetadata object containing all the available metadata. (also stored in the metadata attribute of the OleFileIO object)

new in version 0.25

get_rootentry_name()
Return root entry name. Should usually be 'Root Entry' or 'R' in most implementations.

get_size(*filename*)
Return size of a stream in the OLE container, in bytes.

Parameters **filename** – path of stream in storage tree (see openstream for syntax)

Returns size in bytes (long integer)

Raises

- **IOError** – if file not found
- **TypeError** – if this is not a stream.

get_type(*filename*)
Test if given filename exists as a stream or a storage in the OLE container, and return its type.

Parameters **`filename`** – path of stream in storage tree. (see openstream for syntax)

Returns

False if object does not exist, its entry type (>0) otherwise:

- STGTY_STREAM: a stream
- STGTY_STORAGE: a storage
- STGTY_ROOT: the root entry

`getclsid`(*filename*)
Return clsid of a stream/storage.

Parameters **`filename`** – path of stream/storage in storage tree. (see openstream for syntax)

Returns Empty string if clsid is null, a printable representation of the clsid otherwise

new in version 0.44

`getctime`(*filename*)
Return creation time of a stream/storage.

Parameters **`filename`** – path of stream/storage in storage tree. (see openstream for syntax)

Returns None if creation time is null, a python datetime object otherwise (UTC timezone)

new in version 0.26

`getmtime`(*filename*)
Return modification time of a stream/storage.

Parameters **`filename`** – path of stream/storage in storage tree. (see openstream for syntax)

Returns None if modification time is null, a python datetime object otherwise (UTC timezone)

new in version 0.26

`getproperties`(*filename*, *convert_time=False*, *no_conversion=None*)
Return properties described in substream.

Parameters

- **`filename`** – path of stream in storage tree (see openstream for syntax)
- **`convert_time`** – bool, if True timestamps will be converted to Python datetime
- **`no_conversion`** – None or list of int, timestamps not to be converted (for example total editing time is not a real timestamp)

Returns a dictionary of values indexed by id (integer)

`getsect`(*sect*)
Read given sector from file on disk.

Parameters **`sect`** – int, sector index

Returns a string containing the sector data.

`listdir`(*streams=True*, *storages=False*)
Return a list of streams and/or storages stored in this file

Parameters

- **`streams`** – bool, include streams if True (True by default) - new in v0.26
- **`storages`** – bool, include storages if True (False by default) - new in v0.26 (note: the root storage is never included)

Returns list of stream and/or storage paths

loaddirectory (*sect*)
Load the directory.

Parameters **sect** – sector index of directory stream.

loadfat (*header*)
Load the FAT table.

loadfat_sect (*sect*)
Adds the indexes of the given sector to the FAT

Parameters **sect** – string containing the first FAT sector, or array of long integers

Returns index of last FAT sector.

loadminifat ()
Load the MiniFAT table.

open (*filename*, *write_mode=False*)
Open an OLE2 file in read-only or read/write mode. Read and parse the header, FAT and directory.

Parameters

- **filename** – string-like or file-like object, OLE file to parse
  - if filename is a string smaller than 1536 bytes, it is the path of the file to open. (bytes or unicode string)
  - if filename is a string longer than 1535 bytes, it is parsed as the content of an OLE file in memory. (bytes type only)
  - if filename is a file-like object (with read, seek and tell methods), it is parsed as-is.
- **write_mode** – bool, if True the file is opened in read/write mode instead of read-only by default. (ignored if filename is not a path)

openstream (*filename*)
Open a stream as a read-only file object (BytesIO). Note: filename is case-insensitive.

Parameters **filename** – path of stream in storage tree (except root entry), either:

- a string using Unix path syntax, for example: 'storage_1/storage_1.2/stream'
- or a list of storage filenames, path to the desired stream/storage. Example: ['storage_1', 'storage_1.2', 'stream']

Returns file object (read-only)

Raises **IOError** – if filename not found, or if this is not a stream.

parsing_issues = None
list of defects/issues not raised as exceptions: tuples of (exception type, message)

sect2array (*sect*)
convert a sector to an array of 32 bits unsigned integers, swapping bytes on big endian CPUs such as PowerPC (old Macs)

write_sect (*sect*, *data*, *padding='\x00'*)
Write given sector to file on disk.

Parameters

- **sect** – int, sector index
- **data** – bytes, sector data

- **padding** – single byte, padding character if data < sector size

write_stream(*stream_name*, *data*)

Write a stream to disk. For now, it is only possible to replace an existing stream by data of the same size.

Parameters

- **stream_name** – path of stream in storage tree (except root entry), either:
  - a string using Unix path syntax, for example: 'storage_1/storage_1.2/stream'
  - or a list of storage filenames, path to the desired stream/storage. Example: ['storage_1', 'storage_1.2', 'stream']
- **data** – bytes, data to be written, must be the same size as the original stream.

class olefile.**OleMetadata**

class to parse and store metadata from standard properties of OLE files.

Available attributes: codepage, title, subject, author, keywords, comments, template, last_saved_by, revision_number, total_edit_time, last_printed, create_time, last_saved_time, num_pages, num_words, num_chars, thumbnail, creating_application, security, codepage_doc, category, presentation_target, bytes, lines, paragraphs, slides, notes, hidden_slides, mm_clips, scale_crop, heading_pairs, titles_of_parts, manager, company, links_dirty, chars_with_spaces, unused, shared_doc, link_base, hlinks, hlinks_changed, version, dig_sig, content_type, content_status, language, doc_version

Note: an attribute is set to None when not present in the properties of the OLE file.

References for SummaryInformation stream:

- https://msdn.microsoft.com/en-us/library/dd942545.aspx
- https://msdn.microsoft.com/en-us/library/dd925819%28v=office.12%29.aspx
- https://msdn.microsoft.com/en-us/library/windows/desktop/aa380376%28v=vs.85%29.aspx
- https://msdn.microsoft.com/en-us/library/aa372045.aspx
- http://sedna-soft.de/articles/summary-information-stream/
- https://poi.apache.org/apidocs/org/apache/poi/hpsf/SummaryInformation.html

References for DocumentSummaryInformation stream:

- https://msdn.microsoft.com/en-us/library/dd945671%28v=office.12%29.aspx
- https://msdn.microsoft.com/en-us/library/windows/desktop/aa380374%28v=vs.85%29.aspx
- https://poi.apache.org/apidocs/org/apache/poi/hpsf/DocumentSummaryInformation.html

new in version 0.25

Constructor for OleMetadata All attributes are set to None by default

DOCSUM_ATTRIBS = ['codepage_doc', 'category', 'presentation_target', 'bytes', 'lines',

SUMMARY_ATTRIBS = ['codepage', 'title', 'subject', 'author', 'keywords', 'comments', '

dump()

Dump all metadata, for debugging purposes.

parse_properties(*olefile*)

Parse standard properties of an OLE file, from the streams `\x05SummaryInformation` and `\x05DocumentSummaryInformation`, if present. Properties are converted to strings, integers or python datetime objects. If a property is not present, its value is set to None.

olefile.**enable_logging**()
Enable logging for this module (disabled by default). This will set the module-specific logger level to NOTSET, which means the main application controls the actual logging level.

olefile.**MAGIC = '\xd0\xcf\x11\xe0\xa1\xb1\x1a\xe1'**
magic bytes that should be at the beginning of every OLE file:

olefile.**STGTY_EMPTY = 0**
empty directory entry

olefile.**STGTY_STREAM = 2**
element is a stream object

olefile.**STGTY_STORAGE = 1**
element is a storage object

olefile.**STGTY_ROOT = 5**
element is a root storage

olefile.**STGTY_PROPERTY = 4**
element is an IPropertyStorage object

olefile.**STGTY_LOCKBYTES = 3**
element is an ILockBytes object

olefile.**MAXREGSECT = 4294967290**
(-6) maximum SECT

olefile.**DIFSECT = 4294967292**
(-4) denotes a DIFAT sector in a FAT

olefile.**FATSECT = 4294967293**
(-3) denotes a FAT sector in a FAT

olefile.**ENDOFCHAIN = 4294967294**
(-2) end of a virtual stream chain

olefile.**FREESECT = 4294967295**
(-1) unallocated sector

olefile.**MAXREGSID = 4294967290**
(-6) maximum directory entry ID

olefile.**NOSTREAM = 4294967295**
(-1) unallocated directory entry

CHAPTER 9

Frequently Asked Questions

9.1 Can I extract all images from MS OLE2 documents with olefile?

Not directly: images are not always stored the same way, and it also depends on the format.

For example in Powerpoint presentations, you may find a stream named "Pictures" when running "olefile yourfile.ppt". You may extract the stream by using the openstream() method on the OleFileIO object, but you will usually get a binary stream containing several picture files. You may also extract it manually using tools such as SSView (http://www.mitec.cz/ssv.html).

Then the only way I've found so far is to use file carving tools which are able to determine the beginning and the end of each picture in a binary file. These tools are not always easy to use but if you're interested have a look at https://pypi.org/project/hachoir-subfile/ and http://www.forensicswiki.org/wiki/Tools:Data_Recovery#Carving.

If you really need to automate the process then you have to study Microsoft specifications (at http://www.microsoft.com/interop/docs/officebinaryformats.mspx) and find the right way to parse MS Office documents...

A lot of people (including me) would be very interested if you find a solution! ;-)

Python Module Index

o

olefile, 23

Index

C

close() (olefile.OleFileIO method), 25

D

DIFSECT (in module olefile), 29
DOCSUM_ATTRIBS (olefile.OleMetadata attribute), 28
dump() (olefile.OleMetadata method), 28
dumpdirectory() (olefile.OleFileIO method), 25
dumpfat() (olefile.OleFileIO method), 25
dumpsect() (olefile.OleFileIO method), 25

E

enable_logging() (in module olefile), 28
ENDOFCHAIN (in module olefile), 29
exists() (olefile.OleFileIO method), 25

F

FATSECT (in module olefile), 29
FREESECT (in module olefile), 29

G

get_metadata() (olefile.OleFileIO method), 25
get_rootentry_name() (olefile.OleFileIO method), 25
get_size() (olefile.OleFileIO method), 25
get_type() (olefile.OleFileIO method), 25
getclsid() (olefile.OleFileIO method), 26
getctime() (olefile.OleFileIO method), 26
getmtime() (olefile.OleFileIO method), 26
getproperties() (olefile.OleFileIO method), 26
getsect() (olefile.OleFileIO method), 26

I

isOleFile() (in module olefile), 24

L

listdir() (olefile.OleFileIO method), 26
loaddirectory() (olefile.OleFileIO method), 27
loadfat() (olefile.OleFileIO method), 27
loadfat_sect() (olefile.OleFileIO method), 27
loadminifat() (olefile.OleFileIO method), 27

M

MAGIC (in module olefile), 29
MAXREGSECT (in module olefile), 29
MAXREGSID (in module olefile), 29

N

NOSTREAM (in module olefile), 29

O

olefile (module), 23
OleFileIO (class in olefile), 24
OleMetadata (class in olefile), 28
open() (olefile.OleFileIO method), 27
openstream() (olefile.OleFileIO method), 27

P

parse_properties() (olefile.OleMetadata method), 28
parsing_issues (olefile.OleFileIO attribute), 27

S

sect2array() (olefile.OleFileIO method), 27
STGTY_EMPTY (in module olefile), 29
STGTY_LOCKBYTES (in module olefile), 29
STGTY_PROPERTY (in module olefile), 29
STGTY_ROOT (in module olefile), 29
STGTY_STORAGE (in module olefile), 29
STGTY_STREAM (in module olefile), 29
SUMMARY_ATTRIBS (olefile.OleMetadata attribute), 28

W

write_sect() (olefile.OleFileIO method), 27
write_stream() (olefile.OleFileIO method), 28