dfVFS

Release 20221224

unknown

Dec 29, 2022

CONTENTS

dfVFS, or Digital Forensics Virtual File System, provides read-only access to file-system objects from various storage media types and file formats. The goal of dfVFS is to provide a generic interface for accessing file-system objects, for which it uses several back-ends that provide the actual implementation of the various storage media types, volume systems and file systems.

dfVFS originates from the Plaso project and is also based on ideas from the GRR project. It was largely rewritten and made into a stand-alone project to provide more flexibility and allow other projects to make use of the VFS functionality. dfVFS originally was named PyVFS, but that name conflicted with another project.

The source code is available from the project page.

CHAPTER

ONE

GETTING STARTED

To be able to use dfVFS you first need to install it. There are multiple ways to install dfVFS, check the following instructions for more detail.

1.1 Installation instructions

1.1.1 pip

Note that using pip outside virtualenv is not recommended since it ignores your systems package manager. If you aren't comfortable debugging package installation issues, this is not the option for you.

Create and activate a virtualenv:

```
virtualenv dfvfsenv
cd dfvfsenv
source ./bin/activate
```

Upgrade pip and install dfVFS dependencies:

```
pip install --upgrade pip
pip install dfvfs
```

To deactivate the virtualenv run:

```
deactivate
```

1.1.2 Ubuntu 18.04 and 20.04 LTS

To install dfVFS from the GIFT Personal Package Archive (PPA):

```
sudo add-apt-repository ppa:gift/stable
```

Update and install dfVFS:

```
sudo apt-get update
sudo apt-get install python3-dfvfs
```

1.1.3 Windows

The l2tbinaries contains the necessary packages for running dfVFS. l2tbinaries provides the following branches:

- main; branch intended for the "packaged release" of dfVFS and dependencies;
- dev; branch intended for the "development release" of dfVFS;
- testing; branch intended for testing newly created packages.

The l2tdevtools project provides an update script to ease the process of keeping the dependencies up to date.

The script requires pywin32 and Python WMI.

To install the release versions of the dependencies run:

```
set PYTHONPATH=.

C:\Python3\python.exe tools\update.py --preset dfvfs
```

CHAPTER

TWO

CODE SNIPPETS

This page contains a wide range of code snippets for using dfVFS.

2.1 Working with storage media images and devices

Assume we have a QCOW2 storage media image:

```
qcowmount image.qcow2 fuse/
```

Containing the following partition table:

```
mmls fuse/qcow1
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  Meta    0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000000   0000000062   0000000063   Unallocated
02:  00:00   0000000063   0016755794   0016755732   NTFS (0x07)
03:  -----   0016755795   0016777215   0000021421   Unallocated
```

2.1.1 Exposing a volume as a data range

To create a file-like object of the volume stored in partition 1 (slot `00:00`).

```
from dfvfs.lib import definitions
from dfvfs.path import factory
from dfvfs.resolver import resolver

location = 'image.qcow2'
range_offset = 63 * 512
range_size = 16755732 * 512

os_path_spec = factory.Factory.NewPathSpec(definitions.TYPE_INDICATOR_OS,␣
↪location=location)
qcow_path_spec = factory.Factory.NewPathSpec(definitions.TYPE_INDICATOR_QCOW, parent=os_
↪path_spec)
data_range_path_spec = factory.Factory.NewPathSpec(definitions.TYPE_INDICATOR_DATA_RANGE,
```

(continues on next page)

(continued from previous page)

```
↪ range_offset=range_offset, range_size=range_size, parent=qcow_path_spec)

file_object = resolver.Resolver.OpenFileObject(data_range_path_spec)
```

2.1.2 Traversing a file system

To retrieve a file entry of the root directory:

```
from dfvfs.lib import definitions
from dfvfs.path import factory
from dfvfs.resolver import resolver

location = 'image.qcow2'

os_path_spec = factory.Factory.NewPathSpec(definitions.TYPE_INDICATOR_OS,␣
↪location=location)
qcow_path_spec = factory.Factory.NewPathSpec(definitions.TYPE_INDICATOR_QCOW, parent=os_
↪path_spec)
tsk_partition_path_spec = factory.Factory.NewPathSpec(definitions.TYPE_INDICATOR_TSK_
↪PARTITION, location='/p1', parent=qcow_path_spec)
tsk_path_spec = factory.Factory.NewPathSpec(definitions.TYPE_INDICATOR_TSK, location='/',
↪ parent=tsk_partition_path_spec)

file_entry = resolver.Resolver.OpenFileEntry(tsk_path_spec)
```

To iterate over the sub file entries in the root directory:

```
for sub_file_entry in file_entry.sub_file_entries:
  print(sub_file_entry.name)
```

2.1.3 Retrieving an inode number

```
from dfvfs.lib import definitions
from dfvfs.path import factory
from dfvfs.resolver import resolver
from dfvfs.vfs import attribute

image_location = 'image.E01'
file_location = '/Users/MyUser/AppData/Local/Microsoft/Office/15.0/OfficeFileCache/
↪MyFile.txt'

os_path_spec = factory.Factory.NewPathSpec(definitions.TYPE_INDICATOR_OS, location=image_
↪location)
ewf_path_spec = factory.Factory.NewPathSpec(definitions.TYPE_INDICATOR_EWF, parent=os_
↪path_spec)
tsk_path_spec = factory.Factory.NewPathSpec(definitions.TYPE_INDICATOR_TSK,␣
↪location=file_location, parent=ewf_path_spec)

file_entry = resolver.Resolver.OpenFileEntry(tsk_path_spec)
```

(continues on next page)

(continued from previous page)

```
for file_entry_attribute in file_entry.attributes:
  if isinstance(file_entry_attribute, attribute.StatAttribute):
    print(f'Inode number: {file_entry_attribute.inode_number:d}')
```

CHAPTER

THREE

PATH SPECIFICATIONS

3.1 Terminology

In dfVFS a path specification is defines the location of a file system entry or data stream. It is comparable with the path on an operating system with the difference that the dfVFS path specification includes information about its parents, such a the volume system of the file system.

3.1.1 System-level path specification

A "system-level path specification" is a path specification that can be resolved by the operating system; typically TYPE_INDICATOR_OS or equivalent.

3.2 Type indicators

The dfVFS path specification type indicators are defined in:

```
dfvfs/lib/definitions.py
```

In your code use the type indicator as defined by dfVFS and not its value, in case it changes. The following is a list of type indicators as available in version 20220120.

3.3 Addressing attributes

All types, with the exception of the operating system type, require a parent path specification addressing attribute.

3.3.1 The APFS file system type

The APFS type (TYPE_INDICATOR_APFS) is a type that addresses files stored within an Apple file system (APFS).

3.3.2 The APFS container volume system type

The APFS container type (TYPE_INDICATOR_APFS_CONTAINER) is a type that addresses volumes stored within a Apple file system (APFS) container.

3.3.3 The BDE volume system type

The BDE type (TYPE_INDICATOR_BDE) is a type that addresses volumes stored within a BitLocker encrypted volume.

Note that it is recommended to use the credential manager instead of providing decryption keys (credentials) in a path specification.

3.3.4 The compressed stream type

The compressed stream type (TYPE_INDICATOR_COMPRESSED_STREAM) is an internal type that defines the following addressing attributes:

3.3.5 The cpio archive file type

The cpio type (TYPE_INDICATOR_CPIO) is a type that addresses files stored within the cpio archive file format.

3.3.6 The CS volume system type

The CS type (TYPE_INDICATOR_CS) is a type that addresses volumes stored within a Core Storage (CS) volume system.

Note that it is recommended to use the credential manager instead of providing decryption keys (credentials) in a path specification.

3.3.7 The data range type

The data range type (TYPE_INDICATOR_DATA_RANGE) is an internal type that defines the following addressing attributes:

3.3.8 The encoded stream type

The encoded stream type (TYPE_INDICATOR_ENCODED_STREAM) is an internal type that defines the following addressing attributes:

3.3.9 The encrypted stream type

The encrypted stream type (TYPE_INDICATOR_ENCRYPTED_STREAM) is an internal type that defines the following addressing attributes:

Note that it is recommended to use the credential manager instead of providing decryption keys (credentials) in a path specification.

3.3.10 The EWF storage media image type

The EWF type (TYPE_INDICATOR_EWF) is a type that addresses storage media images stored within the Expert Witness (Compression) Format.

Note that at the moment this type is not addressable as a file system.

Note that at the moment L01 or Lx01 files are not supported.

3.3.11 The EXT file system type

The EXT type (TYPE_INDICATOR_EXT) is a type that addresses files stored within a Extended file system (ext).

3.3.12 The fake file system type

The FAKE type (TYPE_INDICATOR_FAKE) is a virtual file system intended for testing purposes.

3.3.13 The gzip file type

The GZIP type (TYPE_INDICATOR_GZIP) is a type that addresses data stored within the gzip compressed stream file format.

3.3.14 The LUKSDE volume system type

The LUKSDE type (TYPE_INDICATOR_LUKSDE) is a type that addresses volumes stored within a LUKS encrypted volume.

3.3.15 The LVM volume system type

The LVM type (TYPE_INDICATOR_LVM) is a type that addresses volumes stored within a Logical Volume Manager (LVM) volume system.

3.3.16 The mount type

The MOUNT type (TYPE_INDICATOR_MOUNT) is a type that defines a mount point within dfVFS. Also see the mount point manager.

3.3.17 The NTFS file system type

The NTFS type (TYPE_INDICATOR_NTFS) is a type that addresses files stored within a Windows NT file system (NTFS).

3.3.18 The operating system type

The OS type (TYPE_INDICATOR_OS) is a type that addresses files stored within an operating system.

3.3.19 The QCOW storage media image type

The QCOW type (TYPE_INDICATOR_QCOW) is a type that addresses storage media images stored within the QCOW image format, version 1, 2 and 3.

Note that at the moment this type is not addressable as a file system.

3.3.20 The RAW storage media image type

The RAW storage media image type (TYPE_INDICATOR_RAW) is a type that addresses storage media images stored within the RAW image format.

Note that at the moment this type is not addressable as a file system.

3.3.21 The SQlite blob file type

The SQlite blob type (TYPE_INDICATOR_SQLITE_BLOB) is a type that addresses files stored within a blob within a SQLite file.

3.3.22 The tar archive file type

The TAR type (TYPE_INDICATOR_TAR) is a type that addresses files stored within the tar archive file format.

Note that to access e.g. a .tar.gz the a path specification of type TAR should be stacked on top of one of type GZIP.

3.3.23 The SleuthKit file system type

The TSK type (TYPE_INDICATOR_TSK) is a type that addresses files stored within a SleuthKit supported file system.

3.3.24 The SleuthKit volume system type

The TSK_PARTITION type (TYPE_INDICATOR_TSK_PARTITION) is a type that addresses volumes stored within a SleuthKit supported volume system, which largely consists of support for the APM, GPT and MBR partitioning systems.

3.3.25 The VHD storage media image type

The VHDI type (TYPE_INDICATOR_VHDI) is a type that addresses storage media images stored within the Virtual Hard Disk Image format.

Note that at the moment this type is not addressable as a file system.

3.3.26 The VMDK storage media image type

The VMDK type (TYPE_INDICATOR_VMDK) is a type that addresses storage media images stored within the VMWare Virtual Disk Format.

Note that at the moment this type is not addressable as a file system.

3.3.27 The VSS volume system type

The VSHADOW type (TYPE_INDICATOR_VSHADOW) is a type that addresses volumes stored within the Volume Shadow Snapshots (VSS).

3.3.28 The zip archive file type

The ZIP type (TYPE_INDICATOR_ZIP) is a type that addresses files stored within the zip archive file format.

CHAPTER

FOUR

DEVELOPER DOCUMENTATION

dfVFS uses a development process similar to that of Plaso, therefore see Plaso's Developers Guide.

4.1 Introduction

dfVFS was developed to provide a generic means for Plaso to access files via the operating system and those contained in storage media image formats.

In dfVFS terminology:

- a data stream encapsulates the contents of a file, an NTFS ADS or HFS resource fork.
- a file entry encapsulates the file system metadata related to files, directories, symbolic links or equivalent file system related structures.
- a file system encapsulates the hierarchy of file entries within a single volume.
- a volume is part of a volume system and often contains a file system (e.g. NTFS) or a sub volume system (e.g. VSS).
- a volume system encapsulates one or more volumes.

dfVFS separates the concerns relating:

- addressing of file entries, such as the file path, with type specific path specification;
- file-based Input/Output (IO), with type specific basic file-like objects;
- traversing file entries and metadata, with type specific virtual file system and/or file entry;
- format detection, with a format analyzer with type specific helpers.

4.1.1 The path specification

The path specification is equivalent to the path on most operating systems. Though the actual format of the path can be operating systems specific, e.g. Linux uses /home/myuser/myfile.txt and Windows C:\Users\MyUser\MyFile.txt.

In dfVFS the path specification is defined as a container with a *type indicator* with zero or more *addressing attributes*. One of the more common addressing attributes is e.g. location with for the operating system path specification would contain the operating system specific path, e.g. on Windows location=C:\Users\MyUser\MyFile.txt.

Path specifications can be stacked to access files in storage media images and other container formats e.g. archive files like ZIP. For this the path specification can define a *parent* path specification. Note that parent here refers to the path specification and not the parent in the path segment e.g. defined by the location addressing attribute.

Next follows a pseudo code implementation of a stacked path specification that defines the location of C:\Users\MyUser\MyFile.txt on the first partition of QCOW version 2 image.

```
type=OS, location=/home/myuser/myimages/image.qcow2
type=QCOW
type=TSK_PARTITION, location=/p1
type=TSK, inode=128, location=/Users/MyUser/MyFile.txt
```

Note that the parent is not explicitly defined in the example but the parent is defined implicit by the preceding path specification. E.g. OS is the parent of the QCOW path specification type and QCOW the parent of the TSK_PARTITION path specification type.

Also note that though the location of the file within the Windows operating system would be C:\Users\MyUser\MyFile.txt it is defined in the the TSK path specification type as /Users/MyUser/MyFile.txt. dfVFS defines a *Windows path resolver helper* to map Windows paths to a location within images.

The inode addressing attribute in the TSK path specification type is used for faster access to the file within the TSK virtual file system implementation.

TODO

4.1.2 The path specification resolver

The path specification resolver is used to resolve a path specification to a file-like object or virtual file system object.

The path specification resolver uses a resolver context to cache previously resolved file-like object or virtual file system object. This cache uses a weak reference so code interacting with the resolver should maintain a reference if such an object is still used.

4.1.3 The mount point manager

The mount point manager can be used to globally (within the same process space) define mount points used by path specifications of type MOUNT. The same can be accomplished with a resolver context, but note that specified mount points are local to the specific context.

4.1.4 The credentials manager

TODO

4.1.5 Additional helpers

Besides the previously mentioned objects dfVFS also provides several helper objects.

4.2 Helpers

dfVFS currently provides the following helper objects:

- Data slice interface for file-like objects
- Fake file system builder
- Source scanner
- Volume scanner
- File system searcher

- Windows path resolver helper

4.2.1 Data slice interface for file-like objects

The data slice interface for file-like objects provides a wrapper for dfVFS FileIO objects, so that they can be interacted with as data slices. The data slice interface is for example used in Plaso's PE parser.

To create a data slice from a file-like object.

```
from dfvfs.helpers import data_slice

...
slice_object = DataSlice(file_object)
```

To use:

```
signature = slice_object[0:4]
```

4.2.2 Fake file system builder

The fake file system builder is intended for testing purposes. It provides helper functions that take care of setting up a dfVFS fake file system.

To create a fake file system with a single file `/testfile`.

```
from dfvfs.helpers import fake_file_system_builder

file_system_builder = fake_file_system_builder.FakeFileSystemBuilder()
file_system_builder.AddFile('/testfile', b'data')
```

4.2.3 Source scanner

The source scanner was originally created for Plaso tools that deal with storage media devices and images. However it is also used by the dfVFS volume scanner.

The source scanner can be used to analyze source (or input) data, which for Plaso can be an individual file, a directory or a storage media device or image.

The source scanner tries to determine what input we are dealing with:

- a file that contains a storage media image;
- a device file of a storage media image device;
- a regular file or directory.

The source scanner scans for different types of elements:

- supported types of storage media images;
- supported types of volume systems;
- supported types of file systems.

These elements are represented as source scan nodes.

The source scanner uses the source scanner context to keep track of the nodea and user provided context information, such as:

- which partition to default to;
- which VSS stores to default to.

An example of how to use the source scanner can be found in the source analyzer script.

4.2.4 Volume scanner

The volume scanner is an extension of the source analyzer that looks for volumes that contain dfVFS supported volume and file systems. It is intended to help with programmatically handled various types of volume and file systems so the application that uses it can focus on reading file it is interested in.

TODO: add more information about VolumeScannerMediator and VolumeScannerOptions

Examples of how to use the volume scanner can be found in the list file entries and the recursive hasher scripts.

Windows volume scanner

The Windows volume scanner is a variant of the volume scanner that looks for volumes that contain an installation of the Windows operating system.

An example of how to use the source scanner can be found in the WinReg-KB.

4.2.5 File system searcher

The file system searcher was originally created for event extraction with collection filters in Plaso.

TODO: add more information about FindSpec

TODO: add example

4.2.6 Windows path resolver helper

The Windows path resolver helper can be used to resolve various forms Windows paths, e.g. below are several forms of path definitions found in the Windows Registry:

The Windows path resolver helper can be found in:

```
dfvfs/helpers/windows_path_resolver.py
```

TODO: add example

Notes

TODO add more description here

4.3 Adding a new VFS type

Multiple steps are involved in adding a new dfVFS type:

1. Adding a type indicator
2. Adding a path specification
3. Adding a virtual file system and file entry implementation
4. Adding a file IO implementation
5. Adding a path specification resolver helper
6. Adding a format analyzer helper

4.3.1 Adding a type indicator

The type indicators are stored in:

```
dfvfs/lib/definitions.py
```

Add a new type indicator:

```
TYPE_INDICATOR_MY = 'MY'
```

4.3.2 Adding a path specification

The path specifications are stored in:

```
dfvfs/path/
```

Create a new path specification:

```
dfvfs/path/my_path_specification.py
```

Implement a path specification class based on the PathSpec interface class:

```
from dfvfs.lib import definitions
from dfvfs.path import factory
from dfvfs.path import path_spec

class MyPathSpec(path_spec.PathSpec):
  """Class that implements the my path specification."""

  TYPE_INDICATOR = definitions.TYPE_INDICATOR_MY
  ...
```

A path specification class largely consists of:

- an object initialization method (`__init__`) that sets the class attributes;
- a `comparable` property that returns a comparable string form of the path specification.

The comparable string form normally consist of multiple (newline terminated) lines in the form:

```
type: TYPE_INDICATOR, ADDRESSING_ATTRIBUTES
```

where every line represent an individual path specification.

Make sure the path specification will register with the factory on import.

```
factory.Factory.RegisterPathSpec(MyPathSpec)
```

To have the path specification loaded when dfvfs is used, make sure to include in:

```
dfvfs/path/__init__.py
```

```
from dfvfs.path import my_path_specification
```

4.3.3 Adding a virtual file system and file entry implementation

The virtual file systems and file entries are stored in:

```
dfvfs/vfs/
```

The file system and file entry are co-dependent.

Adding a virtual file system implementation

Create a new file system and file entry:

```
dfvfs/path/my_file_system.py
```

Implement a file system class based on the FileSystem interface class:

```
from dfvfs.lib import definitions
from dfvfs.path import my_path_spec
from dfvfs.vfs import file_system
from dfvfs.vfs import my_file_entry

class MyFileSystem(file_system.FileSystem):
  """Class that implements my file system object."""

  TYPE_INDICATOR = definitions.TYPE_INDICATOR_MY
  ...
```

A file system class largely consists of:

- A class constant to define the path (segment) separator used by the file system (`PATH_SEPARATOR`);
- A method to determine if a file entry for a specific path specification exists (`FileEntryExistsByPathSpec`);
- A method to retrieve if a file entry for a specific path specification (`GetFileEntryByPathSpec`);
- A method to retrieve the root file entry (`GetRootFileEntry`).

Adding a virtual file entry implementation

Create a new file system and file entry:

```
dfvfs/path/my_file_entry.py
```

Implement a file entry class based on the FileEntry interface class and a directory class based on the Directory interface class:

```
from dfvfs.lib import definitions
from dfvfs.file_io import fake_file_io
from dfvfs.path import fake_path_spec
from dfvfs.vfs import file_entry

class MyDirectory(file_entry.Directory):
  """Class that implements my directory object."""
  ...

class MyFileEntry(file_entry.FileEntry):
  """Class that implements my file entry object."""

  TYPE_INDICATOR = definitions.TYPE_INDICATOR_MY
  ...
```

A directory class largely consists of:

- A protected method to generate path specification of directory entries for a specific directory path specification (`_EntriesGenerator`).

A file entry class largely consists of:

- (`_GetDirectory`);
- (`_GetStat`);
- a `name` property that ...
- a `sub_file_entries` property that ...
- (`GetFileObject`);
- (`GetParentFileEntry`).

TODO describe a bit more what should be in the class.

4.3.4 Adding a file IO implementation

The file IO (or file-like) objects are stored in:

```
dfvfs/file_io/
```

Create a new file IO object:

```
dfvfs/file_io/my_file_io.py
```

Implement a path specification class based on the FileIO interface class:

```
class MyFile(file_io.FileIO):
  """Class that implements my file-like object."""

  TYPE_INDICATOR = definitions.TYPE_INDICATOR_MY
  ...
```

A file IO class largely consists of:

- (open);
- (close);
- (read);
- (seek);
- (get_offset);
- (get_size);

TODO describe a bit more what should be in the class.

dfVFS does not implement `write` or any of the `readline` functions in its file-like object. Also `tell` is implemented in the FileIO interface as an alias of `get_offset`.

4.3.5 Adding a path specification resolver helper

The path specification resolver helpers are stored in:

```
dfvfs/resolver/
```

Create a new resolver helper:

```
dfvfs/resolver/my_resolver_helper.py
```

Implement a resolver helper class based on the ResolverHelper interface class:

```
from dfvfs.lib import definitions
from dfvfs.resolver import resolver_helper

class MyResolverHelper(resolver_helper.ResolverHelper):
  """Class that implements my resolver helper."""

  TYPE_INDICATOR = definitions.TYPE_INDICATOR_MY
  ...
```

A resolver helper class largely consists of:

- (NewFileObject);
- (NewFileSystem).

TODO describe a bit more what should be in the class.

Make sure the resolver helper will register with the resolver on import.

```
resolver.Resolver.RegisterHelper(MyResolverHelper())
```

To have the resolver helper loaded when the resolver is used, make sure to include in:

```
dfvfs/resolver/__init__.py
```

```
from dfvfs.resolver import my_resolver_helper
```

4.3.6 Adding a format analyzer helper

The format analyzer helpers are stored in:

```
dfvfs/analyzer/
```

Create a new analyzer helper:

```
dfvfs/analyzer/my_analyzer_helper.py
```

Implement an analyzer helper class based on the AnalyzerHelper interface class:

```
from dfvfs.analyzer import analyzer_helper
from dfvfs.lib import definitions

class MyAnalyzerHelper(resolver_helper.AnalyzerHelper):
  """Class that implements my analyzer helper."""

  TYPE_INDICATOR = definitions.TYPE_INDICATOR_MY
  ...
```

An analyzer helper class largely consists of:

- A FORMAT_CATEGORIES definition;
- (`GetFormatSpecification`)

TODO describe a bit more what should be in the class.

Make sure the analyzer helper will register with the analyzer on import.

```
analyzer.Analyzer.RegisterHelper(MyAnalyzerHelper())
```

To have the analyzer helper loaded when the analyzer is used, make sure to include in:

```
dfvfs/analyzer/__init__.py
```

```
from dfvfs.analyzer import my_analyzer_helper
```

4.4 Testing

4.4.1 Unit tests

dfVFS comes with unit tests tests. These tests are stored in the `tests` subdirectory.

To run the unit tests:

```
PYTHONPATH=. python run_tests.py
```

Or on Windows:

```
set PYTHONPATH=.
C:\Python3\python.exe run_tests.py
```

If you're running git on Windows make sure you have autocrlf turned off otherwise the tests using the test text files will fail.

```
git config --global core.autocrlf false
```

CHAPTER

FIVE

SUPPORTED FORMATS

The information below is based of version 20221207

5.1 Storage media types

- EWF (EWF-E01, EWF-Ex01, EWF-S01) (Requires: libewf/pyewf)
- Mac OS disk image (Requires: libmodi/pymodi)
  - Sparse bundle disk image
  - Sparse disk image
  - Universal Disk Image Format (UDIF) image
- Parallels Hard Disk image format version 2 (Requires: libphdi/pyphdi)
- QCOW version 1, 2, 3 (Requires: libqcow/pyqcow)
  - currently no differential image support
- Storage Media device (Requires: libsmdev/pysmdev)
- (split) Storage Media RAW (Requires: libsmraw/pysmraw)
- VHD and VHDX (Requires: libvhdi/pyvhdi)
- VMDK (Requires: libvmdk/pyvmdk)
  - currently no differential image support

5.2 Volume systems

- Apple Partition Map (APM) (Requires: libtsk/pytsk)
- Apple File System (APFS) container version 2 (Requires: libfsapfs/pyfsapfs)
- BitLocker Disk Encryption (BDE) (Requires: libbde/pybde)
- Core Storage (CS) including FileVault Disk Encryption (FVDE) (or FileVault 2) (Requires: libfvde/pyfvde)
- GPT (Requires: libvsgpt/pyvsgpt or libtsk/pytsk)
- LVM (Requires: libvslvm/pyvslvm)
  - At the moment only single physical volume LVM support
- Linux Unified Key Setup (LUKS) (Requires: libluksde/pyluksde)

- MBR (Requires: libtsk/pytsk)
- Volume Shadow Snapshots (VSS) (Requires: libvshadow/pyvshadow)

5.3 File systems

- Apple File System (APFS) version 2 (Requires: libfsapfs/pyfsapfs)
- ext version 2, 3, 4 (Requires: libfsext/pyfsext or libtsk/pytsk)
- FAT-12, FAT-16, FAT-32 and exFAT (Requires: libtsk/pytsk or libfsfat/pyfsfat)
- HFS+, HFSX (Requires: libfshfs/pyfshfs or libtsk/pytsk)
- ISO-9660 version 1 (Requires: libtsk/pytsk)
- NTFS version 3 (Requires: libfsntfs/pyfsntfs or libtsk/pytsk)
- UFS version 1, 2 (Requires: libtsk/pytsk)
- XFS version 4, 5 (Requires: libfsxfs/pyfsxfs)

5.4 Compressed stream file types

- bzip2
- gzip
- lzma
- xz
- zlib (both zlib+DEFLATE and stand-alone DEFLATE)

5.5 Encoded stream file types

- base16
- base32
- base64

5.6 Encrypted stream file types

- AES-CBC, AES-CFB, AES-ECB, AES-OFB (Requires: Cryptography.io)
- Blowfish (Requires: Cryptography.io)
- DES3 (Requires: Cryptography.io)
- RC4 (Requires: Cryptography.io)

5.7 Archive file types

- cpio
- tar
- zip

5.8 Other file types

- blob stored in SQlite

CHAPTER

SIX

DFVFS PACKAGE

6.1 Subpackages

6.1.1 dfvfs.analyzer package

Submodules

dfvfs.analyzer.analyzer module

The format analyzer.

class dfvfs.analyzer.analyzer.**Analyzer**

Bases: object

Format analyzer.

classmethod DeregisterHelper(*analyzer_helper*)

Deregisters a format analyzer helper.

Parameters
analyzer_helper (AnalyzerHelper) – analyzer helper.

Raises
KeyError – if analyzer helper object is not set for the corresponding type indicator.

classmethod GetArchiveTypeIndicators(*path_spec*, *resolver_context=None*)

Determines if a file contains a supported archive types.

Parameters

- **path_spec** (PathSpec) – path specification.
- **resolver_context** (*Optional[*Context*]*) – resolver context, where None represents the built-in context which is not multi process safe.

Returns
supported format type indicators.

Return type
list[str]

classmethod GetCompressedStreamTypeIndicators(*path_spec*, *resolver_context=None*)

Determines if a file contains a supported compressed stream types.

Parameters

- **path_spec** (PathSpec) – path specification.

- **resolver_context** (*Optional[Context]*) – resolver context, where None represents the built-in context which is not multi process safe.

Returns
supported format type indicators.

Return type
list[str]

classmethod GetFileSystemTypeIndicators(*path_spec*, *resolver_context=None*)

Determines if a file contains a supported file system types.

Parameters

- **path_spec** (PathSpec) – path specification.
- **resolver_context** (*Optional[Context]*) – resolver context, where None represents the built-in context which is not multi process safe.

Returns
supported format type indicators.

Return type
list[str]

classmethod GetStorageMediaImageTypeIndicators(*path_spec*, *resolver_context=None*)

Determines if a file contains a supported storage media image types.

Parameters

- **path_spec** (PathSpec) – path specification.
- **resolver_context** (*Optional[Context]*) – resolver context, where None represents the built-in context which is not multi process safe.

Returns
supported format type indicators.

Return type
list[str]

classmethod GetVolumeSystemTypeIndicators(*path_spec*, *resolver_context=None*)

Determines if a file contains a supported volume system types.

Parameters

- **path_spec** (PathSpec) – path specification.
- **resolver_context** (*Optional[Context]*) – resolver context, where None represents the built-in context which is not multi process safe.

Returns
supported format type indicators.

Return type
list[str]

classmethod RegisterHelper(*analyzer_helper*)

Registers a format analyzer helper.

Parameters
analyzer_helper (AnalyzerHelper) – analyzer helper.

Raises
KeyError – if analyzer helper object is already set for the corresponding type indicator.

dfvfs.analyzer.analyzer_helper module

The analyzer helper interface.

class dfvfs.analyzer.analyzer_helper.**AnalyzerHelper**

Bases: object

Analyzer helper interface.

AnalyzeFileObject(*file_object*)

Retrieves the format specification.

This is the fall through implementation that raises a RuntimeError.

Parameters
file_object (FileIO) – file-like object.

Returns

type indicator if the file-like object contains a supported format
or None otherwise.

Return type
str

Raises
NotSupported – since this is the fall through implementation.

GetFormatSpecification()

Retrieves the format specification.

This is the fall through implementation that returns None.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

IsEnabled()

Determines if the analyzer helper is enabled.

Returns
True if the analyzer helper is enabled.

Return type
bool

property format_categories

format categories, such as archive file or file system.

The format categories are defined in definitions.FORMAT_CATEGORIES.

Type
set[str]

property type_indicator

type indicator.

Type
str

dfvfs.analyzer.apfs_analyzer_helper module

The APFS format analyzer helper implementation.

class dfvfs.analyzer.apfs_analyzer_helper.**APFSAnalyzerHelper**

Bases: *AnalyzerHelper*

APFS analyzer helper.

FORMAT_CATEGORIES = frozenset({4})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

TYPE_INDICATOR = 'APFS'

dfvfs.analyzer.apfs_container_analyzer_helper module

The APFS container format analyzer helper implementation.

class dfvfs.analyzer.apfs_container_analyzer_helper.**APFSContainerAnalyzerHelper**

Bases: *AnalyzerHelper*

APFS container analyzer helper.

FORMAT_CATEGORIES = frozenset({6})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

TYPE_INDICATOR = 'APFS_CONTAINER'

dfvfs.analyzer.bde_analyzer_helper module

The BDE format analyzer helper implementation.

class dfvfs.analyzer.bde_analyzer_helper.**BDEAnalyzerHelper**

Bases: *AnalyzerHelper*

BDE analyzer helper.

FORMAT_CATEGORIES = frozenset({6})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot be defined by a specification object.

Return type

FormatSpecification

TYPE_INDICATOR = 'BDE'

dfvfs.analyzer.bzip2_analyzer_helper module

The BZIP2 format analyzer helper implementation.

class dfvfs.analyzer.bzip2_analyzer_helper.**BZIP2AnalyzerHelper**

Bases: *AnalyzerHelper*

BZIP2 analyzer helper.

FORMAT_CATEGORIES = frozenset({2})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot be defined by a specification object.

Return type

FormatSpecification

TYPE_INDICATOR = 'BZIP2'

dfvfs.analyzer.cpio_analyzer_helper module

The CPIO format analyzer helper implementation.

class dfvfs.analyzer.cpio_analyzer_helper.**CPIOAnalyzerHelper**

Bases: *AnalyzerHelper*

CPIO analyzer helper.

FORMAT_CATEGORIES = frozenset({1})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot be defined by a specification object.

Return type

FormatSpecification

TYPE_INDICATOR = 'CPIO'

dfvfs.analyzer.cs_analyzer_helper module

The Core Storage (CS) format analyzer helper implementation.

class dfvfs.analyzer.cs_analyzer_helper.**CSAnalyzerHelper**

Bases: *AnalyzerHelper*

Core Storage (CS) analyzer helper.

FORMAT_CATEGORIES = frozenset({6})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot be defined by a specification object.

Return type

FormatSpecification

TYPE_INDICATOR = 'CS'

dfvfs.analyzer.ewf_analyzer_helper module

The EWF format analyzer helper implementation.

class dfvfs.analyzer.ewf_analyzer_helper.**EWFAnalyzerHelper**

Bases: *AnalyzerHelper*

EWF analyzer helper.

FORMAT_CATEGORIES = frozenset({1, 5})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot be defined by a specification object.

Return type

FormatSpecification

TYPE_INDICATOR = 'EWF'

dfvfs.analyzer.ext_analyzer_helper module

The EXT format analyzer helper implementation.

class dfvfs.analyzer.ext_analyzer_helper.**EXTAnalyzerHelper**

Bases: *AnalyzerHelper*

EXT analyzer helper.

FORMAT_CATEGORIES = frozenset({4})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

IsEnabled()

Determines if the analyzer helper is enabled.

Returns
True if the analyzer helper is enabled.

Return type
bool

TYPE_INDICATOR = 'EXT'

dfvfs.analyzer.fat_analyzer_helper module

The FAT format analyzer helper implementation.

class dfvfs.analyzer.fat_analyzer_helper.**FATAnalyzerHelper**

Bases: *AnalyzerHelper*

FAT analyzer helper.

FORMAT_CATEGORIES = frozenset({4})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

IsEnabled()

Determines if the analyzer helper is enabled.

Returns
True if the analyzer helper is enabled.

Return type
bool

TYPE_INDICATOR = 'FAT'

dfvfs.analyzer.gpt_analyzer_helper module

The GUID Partition Table (GPT) format analyzer helper implementation.

class dfvfs.analyzer.gpt_analyzer_helper.**GPTAnalyzerHelper**

Bases: *AnalyzerHelper*

GPT analyzer helper.

FORMAT_CATEGORIES = frozenset({6})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

IsEnabled()

Determines if the analyzer helper is enabled.

Returns
True if the analyzer helper is enabled.

Return type
bool

TYPE_INDICATOR = 'GPT'

dfvfs.analyzer.gzip_analyzer_helper module

The GZIP format analyzer helper implementation.

class dfvfs.analyzer.gzip_analyzer_helper.**GzipAnalyzerHelper**

Bases: *AnalyzerHelper*

GZIP analyzer helper.

FORMAT_CATEGORIES = frozenset({2})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

TYPE_INDICATOR = 'GZIP'

dfvfs.analyzer.hfs_analyzer_helper module

The HFS format analyzer helper implementation.

class dfvfs.analyzer.hfs_analyzer_helper.**HFSAnalyzerHelper**

Bases: *AnalyzerHelper*

HFS analyzer helper.

FORMAT_CATEGORIES = frozenset({4})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

IsEnabled()

Determines if the analyzer helper is enabled.

Returns
True if the analyzer helper is enabled.

Return type
bool

TYPE_INDICATOR = 'HFS'

dfvfs.analyzer.luksde_analyzer_helper module

The LUKSDE format analyzer helper implementation.

class dfvfs.analyzer.luksde_analyzer_helper.**LUKSDEAnalyzerHelper**

Bases: *AnalyzerHelper*

LUKSDE analyzer helper.

FORMAT_CATEGORIES = frozenset({6})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

TYPE_INDICATOR = 'LUKSDE'

dfvfs.analyzer.lvm_analyzer_helper module

The Logical Volume Manager (LVM) format analyzer helper implementation.

class dfvfs.analyzer.lvm_analyzer_helper.**LVMAnalyzerHelper**

Bases: *AnalyzerHelper*

LVM analyzer helper.

FORMAT_CATEGORIES = frozenset({6})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

TYPE_INDICATOR = 'LVM'

dfvfs.analyzer.modi_analyzer_helper module

The Mac OS disk image (MODI) format analyzer helper implementation.

class dfvfs.analyzer.modi_analyzer_helper.**MODIAnalyzerHelper**

Bases: *AnalyzerHelper*

Mac OS disk image (MODI) analyzer helper.

FORMAT_CATEGORIES = frozenset({5})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

TYPE_INDICATOR = 'MODI'

dfvfs.analyzer.ntfs_analyzer_helper module

The NTFS format analyzer helper implementation.

class dfvfs.analyzer.ntfs_analyzer_helper.**NTFSAnalyzerHelper**

Bases: *AnalyzerHelper*

NTFS analyzer helper.

FORMAT_CATEGORIES = frozenset({4})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

IsEnabled()

Determines if the analyzer helper is enabled.

Returns
True if the analyzer helper is enabled.

Return type
bool

TYPE_INDICATOR = 'NTFS'

dfvfs.analyzer.phdi_analyzer_helper module

The PHDI format analyzer helper implementation.

class dfvfs.analyzer.phdi_analyzer_helper.**PHDIAnalyzerHelper**

Bases: *AnalyzerHelper*

PHDI analyzer helper.

FORMAT_CATEGORIES = frozenset({5})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

TYPE_INDICATOR = 'PHDI'

dfvfs.analyzer.qcow_analyzer_helper module

The QCOW format analyzer helper implementation.

class dfvfs.analyzer.qcow_analyzer_helper.**QCOWAnalyzerHelper**

Bases: *AnalyzerHelper*

QCOW analyzer helper.

FORMAT_CATEGORIES = frozenset({5})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot be defined by a specification object.

Return type
FormatSpecification

TYPE_INDICATOR = 'QCOW'

dfvfs.analyzer.specification module

The format specification classes.

class dfvfs.analyzer.specification.**FormatSpecification**(*identifier*)

Bases: object

Format specification.

AddNewSignature(*pattern*, *offset=None*)

Adds a signature.

Parameters

- **pattern** (*bytes*) – pattern of the signature. Wildcards or regular pattern (regexp) are not supported.
- **offset** (*Optional[int]*) – offset of the signature, where None indicates the signature has no offset. A positive offset or 0 is relative from the start of the data a negative offset is relative to the end of the data.

class dfvfs.analyzer.specification.**FormatSpecificationStore**

Bases: object

Store for format specifications.

AddNewSpecification(*identifier*)

Adds a new format specification.

Parameters
identifier (*str*) – unique signature identifier for a specification store.

Returns
format specification.

Return type
FormatSpecification

Raises
KeyError – if the store already contains a specification with the same identifier.

AddSpecification(*specification*)

Adds a specification.

Parameters
specification (FormatSpecification) – format specification.

Raises
KeyError – if the store already contains a specification with the same identifier.

GetSpecificationBySignature(*signature_identifier*)

Retrieves a specification mapped to a signature identifier.

Parameters
signature_identifier (*str*) – unique signature identifier for a specification store.

Returns

A format specification or None if the signature
identifier does not exist within the specification store.

Return type
FormatSpecification

property specifications

format specifications.

Type
generator[*FormatSpecification*]

class dfvfs.analyzer.specification.**Signature**(*pattern*, *offset=None*)

Bases: object

Signature of a format specification.

The signature consists of a byte string pattern, an optional offset relative to the start of the data, and a value to indicate if the pattern is bound to the offset.

identifier

unique signature identifier for a specification store.

Type
str

offset

offset of the signature, where None indicates the signature has no offset. A positive offset or 0 is relative to the start of the data a negative offset is relative to the end of the data.

Type
int

pattern

pattern of the signature.

Type
bytes

SetIdentifier(*identifier*)

Sets the identifier of the signature in the specification store.

Parameters
identifier (*str*) – unique signature identifier for a specification store.

dfvfs.analyzer.tar_analyzer_helper module

The TAR format analyzer helper implementation.

class dfvfs.analyzer.tar_analyzer_helper.**TARAnalyzerHelper**

Bases: *AnalyzerHelper*

TAR analyzer helper.

FORMAT_CATEGORIES = frozenset({1})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

TYPE_INDICATOR = 'TAR'

dfvfs.analyzer.tsk_analyzer_helper module

The SleuthKit (TSK) format analyzer helper implementation.

class dfvfs.analyzer.tsk_analyzer_helper.**TSKAnalyzerHelper**

Bases: *AnalyzerHelper*

TSK analyzer helper.

FORMAT_CATEGORIES = frozenset({4})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

TYPE_INDICATOR = 'TSK'

dfvfs.analyzer.tsk_partition_analyzer_helper module

The SleuthKit (TSK) partition format analyzer helper implementation.

class dfvfs.analyzer.tsk_partition_analyzer_helper.**TSKPartitionAnalyzerHelper**

Bases: *AnalyzerHelper*

TSK partition analyzer helper.

AnalyzeFileObject(*file_object*)

Retrieves the format specification.

Parameters
file_object (FileIO) – file-like object.

Returns

type indicator if the file-like object contains a supported format
or None otherwise.

Return type
str

FORMAT_CATEGORIES = frozenset({6})

TYPE_INDICATOR = 'TSK_PARTITION'

dfvfs.analyzer.vhdi_analyzer_helper module

The Virtual Hard Disk image (VHDI) format analyzer helper implementation.

class dfvfs.analyzer.vhdi_analyzer_helper.**VHDIAnalyzerHelper**

Bases: *AnalyzerHelper*

Virtual Hard Disk image (VHDI) analyzer helper.

FORMAT_CATEGORIES = frozenset({5})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

TYPE_INDICATOR = 'VHDI'

dfvfs.analyzer.vmdk_analyzer_helper module

The VMDK format analyzer helper implementation.

class dfvfs.analyzer.vmdk_analyzer_helper.**VMDKAnalyzerHelper**

Bases: *AnalyzerHelper*

VMDK analyzer helper.

FORMAT_CATEGORIES = frozenset({5})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

`TYPE_INDICATOR = 'VMDK'`

dfvfs.analyzer.vshadow_analyzer_helper module

The Volume Shadow Snapshots (VSS) format analyzer helper implementation.

class dfvfs.analyzer.vshadow_analyzer_helper.**VShadowAnalyzerHelper**

Bases: *AnalyzerHelper*

VSS analyzer helper.

`FORMAT_CATEGORIES = frozenset({6})`

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

`TYPE_INDICATOR = 'VSHADOW'`

dfvfs.analyzer.xfs_analyzer_helper module

The XFS format analyzer helper implementation.

class dfvfs.analyzer.xfs_analyzer_helper.**XFSAnalyzerHelper**

Bases: *AnalyzerHelper*

XFS analyzer helper.

`FORMAT_CATEGORIES = frozenset({4})`

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

`TYPE_INDICATOR = 'XFS'`

dfvfs.analyzer.xz_analyzer_helper module

The XZ format analyzer helper implementation.

class dfvfs.analyzer.xz_analyzer_helper.**XZAnalyzerHelper**

Bases: *AnalyzerHelper*

XZ analyzer helper.

FORMAT_CATEGORIES = frozenset({2})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

TYPE_INDICATOR = 'XZ'

dfvfs.analyzer.zip_analyzer_helper module

The ZIP format analyzer helper implementation.

class dfvfs.analyzer.zip_analyzer_helper.**ZipAnalyzerHelper**

Bases: *AnalyzerHelper*

ZIP analyzer helper.

FORMAT_CATEGORIES = frozenset({1})

GetFormatSpecification()

Retrieves the format specification.

Returns

format specification or None if the format cannot
be defined by a specification object.

Return type
FormatSpecification

TYPE_INDICATOR = 'ZIP'

Module contents

Imports for the format analyzer.

6.1.2 dfvfs.compression package

Submodules

dfvfs.compression.bzip2_decompressor module

The BZIP2 decompressor implementation.

class dfvfs.compression.bzip2_decompressor.**BZIP2Decompressor**

Bases: *Decompressor*

BZIP2 decompressor using bz2.

COMPRESSION_METHOD = 'bzip2'

Decompress(*compressed_data*)

Decompresses the compressed data.

Parameters
compressed_data (*bytes*) – compressed data.

Returns
uncompressed data and remaining compressed data.

Return type
tuple(bytes, bytes)

Raises
BackEndError – if the BZIP2 compressed stream cannot be decompressed.

dfvfs.compression.decompressor module

The decompressor interface.

class dfvfs.compression.decompressor.**Decompressor**

Bases: object

Decompressor interface.

abstract Decompress(*compressed_data*)

Decompresses the compressed data.

Parameters
compressed_data (*bytes*) – compressed data.

Returns
uncompressed data and remaining compressed data.

Return type
tuple(bytes, bytes)

dfvfs.compression.manager module

The compression manager.

class dfvfs.compression.manager.**CompressionManager**

Bases: object

Compression manager.

classmethod DeregisterDecompressor(*decompressor*)

Deregisters a decompressor for a specific compression method.

Parameters
decompressor (*type*) – decompressor class.

Raises
KeyError – if the corresponding decompressor is not set.

classmethod GetDecompressor(*compression_method*)

Retrieves the decompressor object for a specific compression method.

Parameters
compression_method (*str*) – compression method identifier.

Returns

decompressor or None if the compression method does
not exists.

Return type
Decompressor

classmethod RegisterDecompressor(*decompressor*)

Registers a decompressor for a specific compression method.

Parameters
decompressor (*type*) – decompressor class.

Raises
KeyError – if the corresponding decompressor is already set.

classmethod RegisterDecompressors(*decompressors*)

Registers decompressors.

The decompressors are identified based on their lower case compression method.

Parameters
decompressors (*list[type]*) – decompressor classes.

Raises
KeyError – if decompressor is already set for the corresponding compression method.

dfvfs.compression.xz_decompressor module

The LZMA and XZ decompressor implementations.

class dfvfs.compression.xz_decompressor.**LZMADecompressor**

Bases: *XZDecompressor*

LZMA decompressor using lzma.

COMPRESSION_METHOD = 'lzma'

class dfvfs.compression.xz_decompressor.**XZDecompressor**

Bases: *Decompressor*

XZ decompressor using lzma.

COMPRESSION_METHOD = 'xz'

Decompress(*compressed_data*)

Decompresses the compressed data.

Parameters
compressed_data (*bytes*) – compressed data.

Returns
uncompressed data and remaining compressed data.

Return type
tuple(bytes, bytes)

Raises
BackEndError – if the XZ compressed stream cannot be decompressed.

dfvfs.compression.zlib_decompressor module

The zlib and DEFLATE decompressor implementations.

class dfvfs.compression.zlib_decompressor.**DeflateDecompressor**

Bases: *ZlibDecompressor*

DEFLATE without zlib data decompressor using zlib.

COMPRESSION_METHOD = 'deflate'

class dfvfs.compression.zlib_decompressor.**ZlibDecompressor**(*window_size=15*)

Bases: *Decompressor*

DEFLATE with zlib data decompressor using zlib.

COMPRESSION_METHOD = 'zlib'

Decompress(*compressed_data*)

Decompresses the compressed data.

Parameters
compressed_data (*bytes*) – compressed data.

Returns
uncompressed data and remaining compressed data.

Return type
tuple(bytes, bytes)

Raises
BackEndError – if the zlib compressed stream cannot be decompressed.

property unused_data

data past the end of the compressed data.

Type
bytes

Module contents

Imports for the compression manager.

6.1.3 dfvfs.credentials package

Submodules

dfvfs.credentials.apfs_credentials module

The Apple File System (APFS) credentials.

class dfvfs.credentials.apfs_credentials.**APFSCredentials**

Bases: *Credentials*

Apple File System (APFS) credentials.

CREDENTIALS = frozenset({'password', 'recovery_password'})

TYPE_INDICATOR = 'APFS_CONTAINER'

dfvfs.credentials.bde_credentials module

The BitLocker Drive Encryption (BDE) credentials.

class dfvfs.credentials.bde_credentials.**BDECredentials**

Bases: *Credentials*

BitLocker Drive Encryption (BDE) credentials.

CREDENTIALS = frozenset({'password', 'recovery_password', 'startup_key'})

TYPE_INDICATOR = 'BDE'

dfvfs.credentials.credentials module

The credentials interface.

class dfvfs.credentials.credentials.**Credentials**

Bases: object

Credentials interface.

property type_indicator

type indicator.

Type
str

dfvfs.credentials.cs_credentials module

The Core Storage (CS) credentials.

class dfvfs.credentials.cs_credentials.CSCredentials

Bases: *Credentials*

Core Storage (CS) credentials.

CREDENTIALS = frozenset({'encrypted_root_plist', 'password', 'recovery_password'})

TYPE_INDICATOR = 'CS'

dfvfs.credentials.encrypted_stream_credentials module

The encrypted stream credentials.

class dfvfs.credentials.encrypted_stream_credentials.EncryptedStreamCredentials

Bases: *Credentials*

Encrypted stream credentials.

CREDENTIALS = frozenset({'cipher_mode', 'initialization_vector', 'key'})

TYPE_INDICATOR = 'ENCRYPTED_STREAM'

dfvfs.credentials.keychain module

The path specification key chain.

The key chain is used to manage credentials for path specifications. E.g. BitLocker Drive Encryption (BDE) encrypted volumes can require a credential (e.g. password) to access the unencrypted data (unlock).

class dfvfs.credentials.keychain.KeyChain

Bases: object

Key chain.

Empty()

Empties the key chain.

ExtractCredentialsFromPathSpec(*path_spec*)

Extracts credentials from a path specification.

Parameters
path_spec (PathSpec) – path specification to extract credentials from.

GetCredential(*path_spec*, *identifier*)

Retrieves a specific credential from the key chain.

Parameters

- **path_spec** (PathSpec) – path specification.
- **identifier** (*str*) – credential identifier.

Returns

credential or None if the credential for the path specification
is not set.

Return type
object

GetCredentials(*path_spec*)

Retrieves all credentials for the path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
credentials for the path specification.

Return type
dict[str,object]

SetCredential(*path_spec*, *identifier*, *data*)

Sets a specific credential for the path specification.

Parameters

- **path_spec** (PathSpec) – path specification.
- **identifier** (*str*) – credential identifier.
- **data** (*object*) – credential data.

Raises
KeyError – if the credential is not supported by the path specification type.

dfvfs.credentials.luksde_credentials module

The LUKS Drive Encryption credentials.

class dfvfs.credentials.luksde_credentials.**LUKSDECredentials**

Bases: *Credentials*

LUKS Drive Encryption credentials.

CREDENTIALS = frozenset({'password'})

TYPE_INDICATOR = 'LUKSDE'

dfvfs.credentials.manager module

The path specification credentials manager.

The credentials manager uses credential (instances of Credentials) to specify which credentials a specific path specification type supports. E.g. in case of BitLocker Drive Encryption (BDE): * password; * recovery password; * startup key; * key data.

class dfvfs.credentials.manager.**CredentialsManager**

Bases: object

Credentials manager.

classmethod DeregisterCredentials(*credentials*)

Deregisters a path specification credentials.

Parameters
credentials (Credentials) – credentials.

Raises
KeyError – if credential object is not set for the corresponding type indicator.

classmethod GetCredentials(*path_spec*)

Retrieves the credentials for a specific path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns

credentials or None if the path specification has no
credentials support.

Return type
Credentials

classmethod RegisterCredentials(*credentials*)

Registers a path specification credentials.

Parameters
credentials (Credentials) – credentials.

Raises
KeyError – if credentials object is already set for the corresponding type indicator.

Module contents

Imports for the credential manager.

6.1.4 dfvfs.encoding package

Submodules

dfvfs.encoding.base16_decoder module

The base16 decoder implementation.

class dfvfs.encoding.base16_decoder.**Base16Decoder**

Bases: *Decoder*

Base16 decoder using base64.

Decode(*encoded_data*)

Decode the encoded data.

Parameters
encoded_data (*byte*) – encoded data.

Returns
decoded data and remaining encoded data.

Return type
tuple(bytes, bytes)

Raises
BackEndError – if the base16 stream cannot be decoded.

ENCODING_METHOD = 'base16'

dfvfs.encoding.base32_decoder module

The base32 decoder implementation.

class dfvfs.encoding.base32_decoder.**Base32Decoder**

Bases: *Decoder*

Base32 decoder using base64.

Decode(*encoded_data*)

Decode the encoded data.

Parameters
encoded_data (*byte*) – encoded data.

Returns
decoded data and remaining encoded data.

Return type
tuple(bytes, bytes)

Raises
BackEndError – if the base32 stream cannot be decoded.

ENCODING_METHOD = 'base32'

dfvfs.encoding.base64_decoder module

The base64 decoder implementation.

class dfvfs.encoding.base64_decoder.**Base64Decoder**

Bases: *Decoder*

Base64 decoder using base64.

Decode(*encoded_data*)

Decode the encoded data.

Parameters
encoded_data (*byte*) – encoded data.

Returns
decoded data and remaining encoded data.

Return type
tuple(bytes, bytes)

Raises
BackEndError – if the base64 stream cannot be decoded.

ENCODING_METHOD = 'base64'

dfvfs.encoding.decoder module

The decoder interface.

class dfvfs.encoding.decoder.**Decoder**

Bases: object

Decoder interface.

abstract Decode(*encoded_data*)

Decodes the encoded data.

Parameters
encoded_data (*byte*) – encoded data.

Returns
decoded data and remaining encoded data.

Return type
tuple(bytes, bytes)

dfvfs.encoding.manager module

The encoding manager.

class dfvfs.encoding.manager.**EncodingManager**

Bases: object

Encoding manager.

classmethod DeregisterDecoder(*decoder*)

Deregisters a decoder for a specific encoding method.

Parameters
decoder (*type*) – decoder class.

Raises
KeyError – if the corresponding decoder is not set.

classmethod GetDecoder(*encoding_method*)

Retrieves the decoder object for a specific encoding method.

Parameters
encoding_method (*str*) – encoding method identifier.

Returns
decoder or None if the encoding method does not exists.

Return type
Decoder

classmethod RegisterDecoder(*decoder*)

Registers a decoder for a specific encoding method.

Parameters
decoder (*type*) – decoder class.

Raises
KeyError – if the corresponding decoder is already set.

classmethod RegisterDecoders(*decoders*)

Registers decoders.

The decoders are identified based on their lower case encoding method.

Parameters
decoders (*list[type]*) – decoder classes.

Raises
KeyError – if decoders is already set for the corresponding encoding method.

Module contents

Imports for the encoding manager.

6.1.5 dfvfs.encryption package

Submodules

dfvfs.encryption.aes_decrypter module

The AES decrypter implementation.

class dfvfs.encryption.aes_decrypter.**AESDecrypter**(*cipher_mode=None*, *initialization_vector=None*, *key=None*, ***kwargs*)

Bases: *CryptographyBlockCipherDecrypter*

AES decrypter using Cryptography.

ENCRYPTION_METHOD = 'aes'

dfvfs.encryption.blowfish_decrypter module

The Blowfish decrypter implementation.

class dfvfs.encryption.blowfish_decrypter.**BlowfishDecrypter**(*cipher_mode=None*, *initialization_vector=None*, *key=None*, ***kwargs*)

Bases: *CryptographyBlockCipherDecrypter*

Blowfish decrypter using Cryptography.

ENCRYPTION_METHOD = 'blowfish'

dfvfs.encryption.decrypter module

The decrypter interface.

class dfvfs.encryption.decrypter.**CryptographyBlockCipherDecrypter**(*algorithm=None*, *cipher_mode=None*, *initialization_vector=None*, ***kwargs*)

Bases: *Decrypter*

Block cipher decrypter using Cryptography.

Decrypt(*encrypted_data*, *finalize=False*)

Decrypts the encrypted data.

Parameters

- **encrypted_data** (*bytes*) – encrypted data.
- **finalize** (*Optional[bool]*) – True if the end of data has been reached and the cipher context should be finalized.

Returns
decrypted data and remaining encrypted data.

Return type
tuple[bytes, bytes]

class dfvfs.encryption.decrypter.**Decrypter**(***kwargs*)

Bases: object

Decrypter interface.

abstract Decrypt(*encrypted_data*, *finalize=False*)

Decrypts the encrypted data.

Parameters

- **encrypted_data** (*bytes*) – encrypted data.
- **finalize** (*Optional[bool]*) – True if the end of data has been reached and the cipher context should be finalized.

Returns
decrypted data and remaining encrypted data.

Return type
tuple[bytes, bytes]

dfvfs.encryption.des3_decrypter module

The triple DES decrypter implementation.

class dfvfs.encryption.des3_decrypter.**DES3Decrypter**(*cipher_mode=None*, *initialization_vector=None*, *key=None*, ***kwargs*)

Bases: *CryptographyBlockCipherDecrypter*

Triple DES decrypter using Cryptography.

ENCRYPTION_METHOD = 'des3'

dfvfs.encryption.manager module

The encryption manager.

class dfvfs.encryption.manager.**EncryptionManager**

Bases: object

Encryption manager.

classmethod DeregisterDecrypter(*decrypter*)

Deregisters a decrypter for a specific encryption method.

Parameters
decrypter (*type*) – decrypter class.

Raises
KeyError – if the corresponding decrypter is not set.

classmethod GetDecrypter(*encryption_method*, ***kwargs*)

Retrieves the decrypter object for a specific encryption method.

Parameters

- **encryption_method** (*str*) – encryption method identifier.
- **kwargs** (*dict*) – keyword arguments depending on the decrypter.

Returns
decrypter or None if the encryption method does not exists.

Return type
Decrypter

Raises
CredentialError – if the necessary credentials are missing.

classmethod RegisterDecrypter(*decrypter*)

Registers a decrypter for a specific encryption method.

Parameters
decrypter (*type*) – decrypter class.

Raises
KeyError – if the corresponding decrypter is already set.

classmethod RegisterDecrypters(*decrypters*)

Registers decrypters.

The decrypters are identified based on their lower case encryption method.

Parameters
decrypters (*list[type]*) – decrypter classes.

Raises
KeyError – if decrypters is already set for the corresponding encryption method.

dfvfs.encryption.rc4_decrypter module

The RC4 decrypter implementation.

class dfvfs.encryption.rc4_decrypter.**RC4Decrypter**(*key=None*, ***kwargs*)

Bases: *Decrypter*

RC4 decrypter using Cryptography.

Decrypt(*encrypted_data*, *finalize=False*)

Decrypts the encrypted data.

Parameters

- **encrypted_data** (*bytes*) – encrypted data.
- **finalize** (*Optional[bool]*) – True if the end of data has been reached and the cipher context should be finalized.

Returns
decrypted data and remaining encrypted data.

Return type
tuple[bytes,bytes]

ENCRYPTION_METHOD = 'rc4'

Module contents

Imports for the encryption manager.

6.1.6 dfvfs.file_io package

Submodules

dfvfs.file_io.apfs_file_io module

The Apple File System (APFS) file-like object implementation.

class dfvfs.file_io.apfs_file_io.**APFSFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object using pyfsapfs.file_entry

`get_offset()`

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

`get_size()`

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

`read(size=None)`

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

`seek(offset, whence=0)`

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.bde_file_io module

The BitLocker Drive Encryption (BDE) file-like object.

class dfvfs.file_io.bde_file_io.**BDEFile**(*resolver_context*, *path_spec*)

Bases: *FileObjectIO*

File input/output (IO) object using pybde.

property is_locked

True if the volume is locked.

Type
bool

dfvfs.file_io.compressed_stream_io module

The compressed stream file-like object implementation.

class dfvfs.file_io.compressed_stream_io.**CompressedStream**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object of a compressed stream.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset in the uncompressed stream.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the uncompressed stream.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional[int]*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.cpio_file_io module

The CPIO extracted file-like object implementation.

class dfvfs.file_io.cpio_file_io.**CPIOFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object using CPIOArchiveFile.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset in the CPIO archived file.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the CPIO archived file.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.

- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.cs_file_io module

The Core Storage (CS) file-like object.

class dfvfs.file_io.cs_file_io.**CSFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object using pyfvde.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

property is_locked

True if the volume is locked.

Type
bool

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.data_range_io module

The data range file-like object.

class dfvfs.file_io.data_range_io.**DataRange**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object that maps an in-file data range.

The data range object allows to expose a single partition within a full disk image as a separate file-like object by mapping the data range (offset and size) of the volume on top of the full disk image.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset in the data range.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the data range.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional[int]*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.encoded_stream_io module

The encoded stream file-like object implementation.

class dfvfs.file_io.encoded_stream_io.**EncodedStream**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object of a encoded stream.

SetDecodedStreamSize(*decoded_stream_size*)

Sets the decoded stream size.

This function is used to set the decoded stream size if it can be determined separately.

Parameters

decoded_stream_size (*int*) – size of the decoded stream in bytes.

Raises

- **IOError** – if the file-like object is already open.
- **OSError** – if the file-like object is already open.
- **ValueError** – if the decoded stream size is invalid.

get_offset()

Retrieves the current offset into the file-like object.

Returns

current offset in the decoded stream.

Return type

int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns

size of the decoded stream.

Return type

int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional[int]*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.encrypted_stream_io module

The encrypted stream file-like object implementation.

class dfvfs.file_io.encrypted_stream_io.**EncryptedStream**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object of a encrypted stream.

SetDecryptedStreamSize(*decrypted_stream_size*)

Sets the decrypted stream size.

This function is used to set the decrypted stream size if it can be determined separately.

Parameters
decrypted_stream_size (*int*) – size of the decrypted stream in bytes.

Raises

- **IOError** – if the file-like object is already open.
- **OSError** – if the file-like object is already open.

- **ValueError** – if the decrypted stream size is invalid.

get_offset()

Retrieves the current offset into the decrypted stream.

Returns
current offset into the decrypted stream.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the decrypted stream.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek.
- **whence** (*Optional[int]*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.ewf_file_io module

The EWF image file-like object.

class dfvfs.file_io.ewf_file_io.**EWFFile**(*resolver_context*, *path_spec*)

Bases: *FileObjectIO*

File input/output (IO) object using pyewf.

get_size()

Retrieves the size of the file-like object.

Returns
size of the RAW storage media image inside the EWF container.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

dfvfs.file_io.ext_file_io module

The Extended File System (EXT) file-like object implementation.

class dfvfs.file_io.ext_file_io.**EXTFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object using pyfsext.file_entry

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.fake_file_io module

The fake file-like object implementation.

class dfvfs.file_io.fake_file_io.**FakeFile**(*resolver_context*, *path_spec*, *file_data*)

Bases: *FileIO*

Fake file input/output (IO) object.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.fat_file_io module

The File Allocation Table (FAT) file-like object implementation.

class dfvfs.file_io.fat_file_io.**FATFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object using pyfsfat.file_entry

`get_offset()`

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

`get_size()`

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

`read(size=None)`

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

`seek(offset, whence=0)`

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.file_io module

The Virtual File System (VFS) file input/output (IO) object interface.

class dfvfs.file_io.file_io.**FileIO**(*resolver_context*, *path_spec*)

Bases: object

VFS file input/output (IO) object interface.

Open(*path_spec=None*, *mode='rb'*)

Opens the file input/output (IO) object defined by path specification.

Parameters

- **path_spec** (*Optional[PathSpec]*) – path specification.
- **mode** (*Optional[str]*) – file access mode.

Raises

- ***AccessError*** – if the access to open the file was denied.
- **IOError** – if the file input/output (IO) object was already opened or the open failed.
- **OSError** – if the file input/output (IO) object was already opened or the open failed.
- ***PathSpecError*** – if the path specification is incorrect.
- **ValueError** – if the path specification or mode is invalid.

__del__()

Cleans up the file input/output (IO) object.

abstract get_offset()

Retrieves the current offset into the file input/output (IO) object.

Returns
current offset into the file input/output (IO) object.

Return type
int

Raises

- **IOError** – if the file input/output (IO)-like object has not been opened.
- **OSError** – if the file input/output (IO)-like object has not been opened.

abstract get_size()

Retrieves the size of the file input/output (IO) object.

Returns
size of the file input/output (IO) object.

Return type
int

Raises

- **IOError** – if the file input/output (IO) object has not been opened.
- **OSError** – if the file input/output (IO) object has not been opened.

abstract read(*size=None*)

Reads a byte string from the file input/output (IO) object.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

abstract seek(*offset*, *whence=0*)

Seeks to an offset within the file input/output (IO) object.

Parameters

- **offset** (*int*) – offset to seek.
- **whence** (*Optional[int]*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

seekable()

Determines if a file input/output (IO) object is seekable.

Returns
True since a file IO object provides a seek method.

Return type
bool

tell()

Retrieves the current offset into the file input/output (IO) object.

dfvfs.file_io.file_object_io module

The file object file input/output (IO) object implementation.

class dfvfs.file_io.file_object_io.**FileObjectIO**(*resolver_context*, *path_spec*)

Bases: *FileIO*

Base class for file object-based file input/output (IO) object.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.gpt_file_io module

The GUID Partition Table (GPT) file-like object implementation.

class dfvfs.file_io.gpt_file_io.**GPTFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object using pyvsgpt.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset in the partition.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the partition.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.

- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.gzip_file_io module

The gzip file-like object.

class dfvfs.file_io.gzip_file_io.**GzipFile**(*resolver_context*, *path_spec*)

Bases: *FileObjectIO*

File input/output (IO) object of a gzip file.

property comments

comments in the gzip file.

Type
list(str)

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

property modification_times

modification times stored in the gzip file.

Type
list(int)

property operating_systems

operating system values stored in the gzip file.

Type
list(int)

property original_filenames

original filenames stored in the gzip file.

Type
list(str)

property uncompressed_data_size

uncompressed data size.

Type
int

dfvfs.file_io.hfs_file_io module

The Hierarchical File System (HFS) file-like object implementation.

class dfvfs.file_io.hfs_file_io.**HFSFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object using pyfshfs.file_entry

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.

- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.luksde_file_io module

The LUKS Drive Encryption file-like object.

class dfvfs.file_io.luksde_file_io.**LUKSDEFile**(*resolver_context*, *path_spec*)

Bases: *FileObjectIO*

File input/output (IO) object using pyluksde.

property is_locked

True if the volume is locked.

Type
bool

dfvfs.file_io.lvm_file_io module

The Logical Volume Manager (LVM) file-like object implementation.

class dfvfs.file_io.lvm_file_io.**LVMFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object using pyvslvm.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.modi_file_io module

The Mac OS disk image file-like object.

class dfvfs.file_io.modi_file_io.**MODIFile**(*resolver_context*, *path_spec*)

Bases: *FileObjectIO*

File input/output (IO) object using pymodi.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

dfvfs.file_io.ntfs_file_io module

The NTFS file-like object implementation.

class dfvfs.file_io.ntfs_file_io.**NTFSFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object using pyfsntfs.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.

- **whence** (*Optional[int]*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.os_file_io module

The operating system file-like object implementation.

class dfvfs.file_io.os_file_io.**OSFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object that uses the operating system.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.phdi_file_io module

The PHDI image file-like object.

class dfvfs.file_io.phdi_file_io.**PHDIFile**(*resolver_context*, *path_spec*)

Bases: *FileObjectIO*

File input/output (IO) object using pyphdi.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

dfvfs.file_io.qcow_file_io module

The QCOW image file-like object.

class dfvfs.file_io.qcow_file_io.**QCOWFile**(*resolver_context*, *path_spec*)

Bases: *FileObjectIO*

File input/output (IO) object using pyqcow.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

dfvfs.file_io.raw_file_io module

The RAW storage media image file-like object implementation.

class dfvfs.file_io.raw_file_io.**RawFile**(*resolver_context*, *path_spec*)

Bases: *FileObjectIO*

File input/output (IO) object using pysmraw.

get_size()

Retrieves the size of the file-like object.

Returns
size of the RAW storage media image.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

dfvfs.file_io.sqlite_blob_file_io module

The SQlite blob file-like object.

class dfvfs.file_io.sqlite_blob_file_io.**SQLiteBlobFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object using sqlite.

GetNumberOfRows()

Retrieves the number of rows of the table.

Returns
number of rows.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.tar_file_io module

The TAR extracted file-like object implementation.

class dfvfs.file_io.tar_file_io.**TARFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object using tarfile.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.

- **whence** (*Optional[int]*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.tsk_file_io module

The SleuthKit (TSK) file-like object implementation.

class dfvfs.file_io.tsk_file_io.**TSKFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object using pytsk3.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.tsk_partition_file_io module

The SleuthKit (TSK) partition file-like object implementation.

class dfvfs.file_io.tsk_partition_file_io.**TSKPartitionFile**(*resolver_context*, *path_spec*)

Bases: *DataRange*

File input/output (IO) object using pytsk3.

dfvfs.file_io.vhdi_file_io module

The Virtual Hard Disk image file-like object.

class dfvfs.file_io.vhdi_file_io.**VHDIFile**(*resolver_context*, *path_spec*)

Bases: *FileObjectIO*

File input/output (IO) object using pyvhdi.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

dfvfs.file_io.vmdk_file_io module

The VMDK image file-like object.

class dfvfs.file_io.vmdk_file_io.**VMDKFile**(*resolver_context*, *path_spec*)

Bases: *FileObjectIO*

File input/output (IO) object using pyvmdk.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

dfvfs.file_io.vshadow_file_io module

The Volume Shadow Snapshots (VSS) file-like object implementation.

class dfvfs.file_io.vshadow_file_io.**VShadowFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object using pyvshadow.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.xfs_file_io module

The X File System (XFS) file-like object implementation.

class dfvfs.file_io.xfs_file_io.**XFSFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object using pyfsxfs.file_entry

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

read(*size=None*)

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

dfvfs.file_io.zip_file_io module

The ZIP extracted file-like object implementation.

class dfvfs.file_io.zip_file_io.**ZipFile**(*resolver_context*, *path_spec*)

Bases: *FileIO*

File input/output (IO) object using zipfile.

`get_offset()`

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

`get_size()`

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

`read(size=None)`

Reads a byte string from the file-like object at the current offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

`seek(offset, whence=0)`

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional(int)*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

Module contents

6.1.7 dfvfs.helpers package

Submodules

dfvfs.helpers.command_line module

Helpers for command line tools.

class dfvfs.helpers.command_line.**CLIInputReader**(*encoding='utf-8'*)

Bases: object

Command line interface input reader interface.

abstract Read()

Reads a string from the input.

Returns
input.

Return type
str

class dfvfs.helpers.command_line.**CLIOutputWriter**(*encoding='utf-8'*)

Bases: object

Command line interface output writer interface.

Flush()

Flushes buffered data to the output.

abstract Write(*string*)

Writes a string to the output.

Parameters
string (*str*) – output.

class dfvfs.helpers.command_line.**CLITabularTableView**(*column_names=None*, *column_sizes=None*)

Bases: object

Command line interface tabular table view.

AddRow(*values*)

Adds a row of values.

Parameters
values (*list[object]*) – values.

Raises
ValueError – if the number of values is out of bounds.

Write(*output_writer*)

Writes the table to output writer.

Parameters
output_writer (CLIOutputWriter) – output writer.

class dfvfs.helpers.command_line.**CLIVolumeScannerMediator**(*input_reader=None*, *output_writer=None*)

Bases: *VolumeScannerMediator*

Command line volume scanner mediator.

GetAPFSVolumeIdentifiers(*volume_system*, *volume_identifiers*)

Retrieves APFS volume identifiers.

This method can be used to prompt the user to provide APFS volume identifiers.

Parameters

- **volume_system** (APFSVolumeSystem) – volume system.
- **volume_identifiers** (*list[str]*) – volume identifiers including prefix.

Returns
selected volume identifiers including prefix or None.

Return type
list[str]

GetLVMVolumeIdentifiers(*volume_system*, *volume_identifiers*)

Retrieves LVM volume identifiers.

This method can be used to prompt the user to provide LVM volume identifiers.

Parameters

- **volume_system** (LVMVolumeSystem) – volume system.
- **volume_identifiers** (*list[str]*) – volume identifiers including prefix.

Returns
selected volume identifiers including prefix or None.

Return type
list[str]

GetPartitionIdentifiers(*volume_system*, *volume_identifiers*)

Retrieves partition identifiers.

This method can be used to prompt the user to provide partition identifiers.

Parameters

- **volume_system** (VolumeSystem) – volume system.
- **volume_identifiers** (*list[str]*) – volume identifiers including prefix.

Returns
selected volume identifiers including prefix or None.

Return type
list[str]

GetVSSStoreIdentifiers(*volume_system*, *volume_identifiers*)

Retrieves VSS store identifiers.

This method can be used to prompt the user to provide VSS store identifiers.

Parameters

- **volume_system** (VShadowVolumeSystem) – volume system.

- **volume_identifiers** (*list[str]*) – volume identifiers including prefix.

Returns
selected volume identifiers including prefix or None.

Return type
list[str]

ParseVolumeIdentifiersString(*volume_identifiers_string*)

Parses a user specified volume identifiers string.

Parameters
volume_identifiers_string (*str*) – user specified volume identifiers. A range of volumes can be defined as: "3..5". Multiple volumes can be defined as: "1,3,5" (a list of comma separated values). Ranges and lists can also be combined as: "1,3..5". The first volume is 1. All volumes can be defined as "all". No volumes can be defined as an empty string or "none".

Returns
volume identifiers with prefix or the string "all".

Return type
list[str]

Raises
ValueError – if the volume identifiers string is invalid.

PrintWarning(*warning*)

Prints a warning.

Parameters
warning (*str*) – warning text.

UnlockEncryptedVolume(*source_scanner_object*, *scan_context*, *locked_scan_node*, *credentials*)

Unlocks an encrypted volume.

This method can be used to prompt the user to provide encrypted volume credentials.

Parameters

- **source_scanner_object** (SourceScanner) – source scanner.
- **scan_context** (SourceScannerContext) – source scanner context.
- **locked_scan_node** (SourceScanNode) – locked scan node.
- **credentials** (Credentials) – credentials supported by the locked scan node.

Returns
True if the volume was unlocked.

Return type
bool

class dfvfs.helpers.command_line.**FileObjectInputReader**(*file_object*, *encoding='utf-8'*)

Bases: *CLIInputReader*

File object command line interface input reader.

This input reader relies on the file-like object having a readline method.

Read()

Reads a string from the input.

Returns
input.

Return type
str

class dfvfs.helpers.command_line.**FileObjectOutputWriter**(*file_object*, *encoding='utf-8'*)

Bases: *CLIOutputWriter*

File object command line interface output writer.

This output writer relies on the file-like object having a write method.

Write(*string*)

Writes a string to the output.

Parameters
string (*str*) – output.

class dfvfs.helpers.command_line.**StdinInputReader**(*encoding='utf-8'*)

Bases: *FileObjectInputReader*

Stdin command line interface input reader.

class dfvfs.helpers.command_line.**StdoutOutputWriter**(*encoding='utf-8'*)

Bases: *CLIOutputWriter*

Stdout command line interface output writer.

Flush()

Flushes buffered data to the output.

Write(*string*)

Writes a string to the output.

Parameters
string (*str*) – output.

dfvfs.helpers.data_slice module

A data slice interface for file-like objects.

class dfvfs.helpers.data_slice.**DataSlice**(*file_object*)

Bases: object

Data slice interface for file-like objects.

__enter__()

Enters a with statement.

__exit__(*unused_type*, *unused_value*, *unused_traceback*)

Exits a with statement.

__getitem__(*key*)

Retrieves a range of file data.

Parameters
key (*int|slice*) – offset or range of offsets to retrieve file data from.

Returns
range of file data.

Return type
bytes

Raises

- **TypeError** – if the type of the key is not supported.
- **ValueError** – if the step value of a slice is not None.

__len__()

Retrieves the file data size.

Returns
file data size.

Return type
int

dfvfs.helpers.fake_file_system_builder module

A builder for fake file systems.

class dfvfs.helpers.fake_file_system_builder.**FakeFileSystemBuilder**

Bases: object

Builder object for fake file systems.

file_system

fake file system.

Type
FakeFileSystem

AddDirectory(*path*)

Adds a directory to the fake file system.

Note that this function will create parent directories if needed.

Parameters
path (*str*) – path of the directory within the fake file system.

Raises
ValueError – if the path is already set.

AddFile(*path*, *file_data*)

Adds a "regular" file to the fake file system.

Note that this function will create parent directories if needed.

Parameters

- **path** (*str*) – path of the file within the fake file system.
- **file_data** (*bytes*) – data of the file.

Raises
ValueError – if the path is already set.

AddFileReadData(*path*, *file_data_path*)

Adds a "regular" file to the fake file system.

Parameters

- **path** (*str*) – path of the file within the fake file system.
- **file_data_path** (*str*) – path of the file to read the file data from.

Raises
ValueError – if the path is already set.

AddSymbolicLink(*path*, *linked_path*)

Adds a symbolic link to the fake file system.

Parameters

- **path** (*str*) – path of the symbolic link within the fake file system.
- **linked_path** (*str*) – path that is linked.

Raises
ValueError – if the path is already set.

dfvfs.helpers.file_system_searcher module

A searcher to find file entries within a file system.

class dfvfs.helpers.file_system_searcher.**FileSystemSearcher**(*file_system*, *mount_point*)

Bases: object

Searcher to find file entries within a file system.

Find(*find_specs=None*)

Searches for matching file entries within the file system.

Parameters
find_specs (*list[FindSpec]*) – find specifications. where None will return all allocated file entries.

Yields
PathSpec – path specification of a matching file entry.

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
file entry or None.

Return type
FileEntry

GetRelativePath(*path_spec*)

Returns the relative path based on a resolved path specification.

The relative path is the location of the upper most path specification. The the location of the mount point is stripped off if relevant.

Parameters
path_spec (PathSpec) – path specification.

Returns

corresponding relative path or None if the relative path could not
be determined.

Return type
str

Raises
PathSpecError – if the path specification is incorrect.

`SplitPath`(*path*)

Splits the path into path segments.

Parameters
path (*str*) – path.

Returns

path segments without the root path segment, which is an
empty string.

Return type
list[str]

`class dfvfs.helpers.file_system_searcher.FindSpec`(*case_sensitive=True*, *file_entry_types=None*, *is_allocated=True*, *location=None*, *location_glob=None*, *location_regex=None*, *location_separator='/'*)

Bases: `object`

Find specification.

`AtLastLocationSegment`(*segment_index*)

Determines if the a location segment is the last one or greater.

Parameters
segment_index (*int*) – index of the location path segment.

Returns
True if at maximum depth, False if not.

Return type
bool

`CompareLocation`(*file_entry*, *mount_point=None*)

Compares a file entry location against the find specification.

Parameters

- **file_entry** (FileEntry) – file entry.
- **mount_point** (*Optional[PathSpec]*) – mount point path specification that refers to the base location of the file system. The mount point is ignored if it is not an OS path specification.

Returns

True if the location of the file entry matches that of the find
specification, False if not or if the find specification has no location defined.

Return type
bool

Raises
ValueError – if mount point is set and is of type OS and the location of the path specification of the file entry falls outside the mount point.

`CompareNameWithLocationSegment`(*file_entry*, *segment_index*)

Compares a file entry name against a find specification location segment.

Parameters

- **file_entry** (FileEntry) – file entry.
- **segment_index** (*int*) – index of the location segment to compare against, where 0 represents the root segment.

Returns

True if the location segment of the file entry matches that of the find specification, False if not or if the find specification has no location defined.

Return type
bool

`CompareTraits`(*file_entry*)

Compares a file entry traits against the find specification.

Parameters
file_entry (FileEntry) – file entry.

Returns

True if the traits of the file entry, such as type, matches the find specification, False otherwise.

Return type
bool

`HasLocation`()

Determines if the find specification has a location defined.

Returns
True if find specification has a location defined, False if not.

Return type
bool

`IsLastLocationSegment`(*segment_index*)

Determines if the a location segment is the last one.

Parameters
segment_index (*int*) – index of the location path segment.

Returns
True if at maximum depth, False if not.

Return type
bool

dfvfs.helpers.source_scanner module

Scanner for source files, directories and devices.

The source scanner tries to determine what input we are dealing with: * a file that contains a storage media image; * a device file of a storage media image device; * a regular file or directory.

The source scanner scans for different types of elements: * supported types of storage media images; * supported types of volume systems; * supported types of file systems.

These elements are represented as source scan nodes.

The source scanner uses the source scanner context to keep track of the nodea and user provided context information, such as: * which partition to default to; * which VSS stores to default to.

class dfvfs.helpers.source_scanner.**SourceScanNode**(*path_spec*)

Bases: object

Source scan node.

credential

credential used to unlock the source scan node.

Type
tuple[str, str]

path_spec

path specification.

Type
PathSpec

parent_node

source scan parent node.

Type
SourceScanNode

scanned

True if the source scan node has been fully scanned.

Type
bool

sub_nodes

source scan sub nodes.

Type
list[*SourceScanNode*]

GetSubNodeByLocation(*location*)

Retrieves a sub scan node based on the location.

Parameters
location (*str*) – location that should match the location of the path specification of a sub scan node.

Returns
sub scan node or None if not available.

Return type
SourceScanNode

`GetUnscannedSubNode()`

Retrieves the first unscanned sub node.

Returns
sub scan node or None if not available.

Return type
SourceScanNode

`IsFileSystem()`

Determines if the scan node represents a file system.

Returns
True if the scan node represents a file system.

Return type
bool

`IsSystemLevel()`

Determines if the scan node has a path specification at system-level.

System-level is an indication used if the path specification is handled by the operating system and should not have a parent.

Returns
True if the scan node has a path specification at system-level.

Return type
bool

`IsVolumeSystem()`

Determines if the scan node represents a volume system.

Returns
True if the scan node represents a volume system.

Return type
bool

`IsVolumeSystemRoot()`

Determines if the scan node represents the root of a volume system.

Returns
True if the scan node represents the root of a volume system.

Return type
bool

`SupportsEncryption()`

Determines if the scan node supports encryption.

Returns
True if the scan node supports encryption.

Return type
bool

`property type_indicator`

path specification type indicator.

Type
str

class dfvfs.helpers.source_scanner.**SourceScanner**(*resolver_context=None*)

Bases: object

Searcher to find volumes within a volume system.

GetVolumeIdentifiers(*volume_system*)

Retrieves the volume identifiers.

Parameters
volume_system (VolumeSystem) – volume system.

Returns
sorted volume identifiers.

Return type
list[str]

Scan(*scan_context*, *auto_recurse=True*, *scan_path_spec=None*)

Scans for supported formats.

Parameters

- **scan_context** (SourceScannerContext) – source scanner context.
- **auto_recurse** (*Optional[bool]*) – True if the scan should automatically recurse as far as possible.
- **scan_path_spec** (*Optional[PathSpec]*) – path specification to indicate where the source scanner should continue scanning, where None indicates the scanner will start with the sources.

Raises
ValueError – if the scan context is invalid.

ScanForFileSystem(*source_path_spec*)

Scans the path specification for a supported file system format.

Parameters
source_path_spec (PathSpec) – source path specification.

Returns

file system path specification or None if no supported file
system type was found.

Return type
PathSpec

Raises
BackEndError – if the source cannot be scanned or more than one file system type is found.

ScanForStorageMediaImage(*source_path_spec*)

Scans the path specification for a supported storage media image format.

Parameters
source_path_spec (PathSpec) – source path specification.

Returns

storage media image path specification or None if no supported
storage media image type was found.

Return type
PathSpec

Raises
BackEndError – if the source cannot be scanned or more than one storage media image type is found.

ScanForVolumeSystem(*source_path_spec*)

Scans the path specification for a supported volume system format.

Parameters
source_path_spec (PathSpec) – source path specification.

Returns

volume system path specification or None if no supported volume
system type was found.

Return type
PathSpec

Raises
BackEndError – if the source cannot be scanned or more than one volume system type is found.

Unlock(*scan_context*, *path_spec*, *credential_identifier*, *credential_data*)

Unlocks a locked scan node e.g. the scan node of an encrypted volume.

Parameters

- **scan_context** (SourceScannerContext) – source scanner context.
- **path_spec** (PathSpec) – path specification of the locked scan node.
- **credential_identifier** (*str*) – credential identifier used to unlock the scan node.
- **credential_data** (*bytes*) – credential data used to unlock the scan node.

Returns
True if the scan node was successfully unlocked.

Return type
bool

Raises

- ***BackEndError*** – if the scan node cannot be unlocked.
- **KeyError** – if the scan node does not exists or is not locked.

class dfvfs.helpers.source_scanner.**SourceScannerContext**

Bases: object

Contextual information for the source scanner.

source_type

type of source.

Type
str

updated

True if the source scanner context has been updated.

Type
bool

AddScanNode(*path_spec*, *parent_scan_node*)

Adds a scan node for a certain path specification.

Parameters

- **path_spec** (PathSpec) – path specification.
- **parent_scan_node** (SourceScanNode) – parent scan node or None.

Returns
scan node.

Return type
SourceScanNode

Raises

- **KeyError** – if the scan node already exists.
- **RuntimeError** – if the parent scan node is not present.

GetRootScanNode()

Retrieves the root scan node.

Returns
scan node or None if not available.

Return type
SourceScanNode

GetScanNode(*path_spec*)

Retrieves a scan node for a certain path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
scan node or None if not available.

Return type
SourceScanNode

GetUnscannedScanNode()

Retrieves the first unscanned scan node.

Returns
scan node or None if not available.

Return type
SourceScanNode

HasFileSystemScanNodes()

Determines if a file system was detected during the scan.

Returns
True if a file system was detected during the scan.

Return type
bool

`HasLockedScanNodes()`

Determines if a locked scan node was detected during the scan.

A locked scan node is e.g. an encrypted volume for which a credential, e.g. password, to unlock the volume is not available.

Returns
True if a locked scan node was detected during the scan.

Return type
bool

`HasScanNode(path_spec)`

Determines if there is a scan node for a certain path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if there is a scan node for the path specification.

Return type
bool

`IsLockedScanNode(path_spec)`

Determines if a scan node is locked.

A locked scan node is e.g. an encrypted volume for which a credential, e.g. password, to unlock the volume is not available.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the scan node is locked.

Return type
bool

`IsSourceTypeDirectory()`

Determines if the source type is a directory.

Returns

True if the source type is a directory, False if not or
None if not set.

Return type
bool

`IsSourceTypeFile()`

Determines if the source type is a file.

Returns
True if the source type is a file, False if not or None if not set.

Return type
bool

`LockScanNode(path_spec)`

Marks a scan node as locked.

Parameters
path_spec (PathSpec) – path specification.

Raises
KeyError – if the scan node does not exists.

OpenSourcePath(*source_path*)

Opens the source path.

Parameters
source_path (*str*) – source path.

RemoveScanNode(*path_spec*)

Removes a scan node of a certain path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
parent scan node or None if not available.

Return type
SourceScanNode

Raises
RuntimeError – if the scan node has sub nodes.

SOURCE_TYPE_DIRECTORY = 'directory'

SOURCE_TYPE_FILE = 'file'

SOURCE_TYPE_STORAGE_MEDIA_DEVICE = 'storage media device'

SOURCE_TYPE_STORAGE_MEDIA_IMAGE = 'storage media image'

SetSourceType(*source_type*)

Sets the source type.

Parameters
source_type (*str*) – source type.

UnlockScanNode(*path_spec*, *credential_identifier*, *credential_data*)

Marks a scan node as unlocked.

Parameters

- **path_spec** (PathSpec) – path specification.
- **credential_identifier** (*str*) – credential identifier used to unlock the scan node.
- **credential_data** (*bytes*) – credential data used to unlock the scan node.

Raises
KeyError – if the scan node does not exists or is not locked.

property locked_scan_nodes

locked scan nodes.

Type
list[*SourceScanNode*]

dfvfs.helpers.text_file module

A text file interface for file-like objects.

class dfvfs.helpers.text_file.**TextFile**(*file_object*, *encoding='utf-8'*, *encoding_errors='strict'*, *end_of_line='\n'*)

Bases: object

Text file interface for file-like objects.

__enter__()

Enters a with statement.

__exit__(*unused_type*, *unused_value*, *unused_traceback*)

Exits a with statement.

__iter__()

Returns a line of text.

Yields
str – line of text.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

readline(*size=None*)

Reads a single line of text.

The functions reads one entire line from the file-like object. A trailing end-of-line indicator (newline by default) is kept in the string (but may be absent when a file ends with an incomplete line). An empty string is returned only when end-of-file is encountered immediately.

Parameters
size (*Optional[int]*) – maximum byte size to read. If present and non-negative, it is a maximum byte count (including the trailing end-of-line) and an incomplete line may be returned.

Returns
line of text.

Return type
str

Raises

- **UnicodeDecodeError** – if a line cannot be decoded and encoding errors is set to strict.
- **ValueError** – if the size is smaller than zero or exceeds the maximum (as defined by _MAXIMUM_READ_BUFFER_SIZE).

readlines(*sizehint=None*)

Reads lines of text.

The function reads until EOF using readline() and return a list containing the lines read.

Parameters
sizehint (*Optional[int]*) – maximum byte size to read. If present, instead of reading up to EOF, whole lines totalling sizehint bytes are read.

Returns
lines of text.

Return type
list[str]

`tell()`

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

dfvfs.helpers.volume_scanner module

Scanner for supported volume and file systems.

`class dfvfs.helpers.volume_scanner.VolumeScanner(mediator=None)`

Bases: `object`

Volume scanner.

`GetBasePathSpecs(source_path, options=None)`

Determines the base path specifications.

Parameters
- **source_path** (*str*) – source path.
- **options** (*Optional[VolumeScannerOptions]*) – volume scanner options. If None the default volume scanner options are used, which are defined in the VolumeScannerOptions class.

Returns
path specifications.

Return type
list[*PathSpec*]

Raises
ScannerError – if the source path does not exists, or if the source path is not a file or directory, or if the format of or within the source file is not supported.

`class dfvfs.helpers.volume_scanner.VolumeScannerMediator`

Bases: `object`

Volume scanner mediator.

`abstract GetAPFSVolumeIdentifiers(volume_system, volume_identifiers)`

Retrieves APFS volume identifiers.

This method can be used to prompt the user to provide APFS volume identifiers.

Parameters
- **volume_system** (APFSVolumeSystem) – volume system.

- **volume_identifiers** (*list[str]*) – volume identifiers including prefix.

Returns
selected volume identifiers including prefix or None.

Return type
list[str]

abstract GetLVMVolumeIdentifiers(*volume_system*, *volume_identifiers*)

Retrieves LVM volume identifiers.

This method can be used to prompt the user to provide LVM volume identifiers.

Parameters

- **volume_system** (LVMVolumeSystem) – volume system.
- **volume_identifiers** (*list[str]*) – volume identifiers including prefix.

Returns
selected volume identifiers including prefix or None.

Return type
list[str]

abstract GetPartitionIdentifiers(*volume_system*, *volume_identifiers*)

Retrieves partition identifiers.

This method can be used to prompt the user to provide partition identifiers.

Parameters

- **volume_system** (TSKVolumeSystem) – volume system.
- **volume_identifiers** (*list[str]*) – volume identifiers including prefix.

Returns
selected volume identifiers including prefix or None.

Return type
list[str]

abstract GetVSSStoreIdentifiers(*volume_system*, *volume_identifiers*)

Retrieves VSS store identifiers.

This method can be used to prompt the user to provide VSS store identifiers.

Parameters

- **volume_system** (VShadowVolumeSystem) – volume system.
- **volume_identifiers** (*list[str]*) – volume identifiers including prefix.

Returns
selected volume identifiers including prefix or None.

Return type
list[str]

GetVolumeIdentifiers(*volume_system*, *volume_identifiers*)

Retrieves volume identifiers.

This method can be used to prompt the user to provide volume identifiers.

Parameters

- **volume_system** (APFSVolumeSystem) – volume system.
- **volume_identifiers** (*list[str]*) – volume identifiers including prefix.

Returns
selected volume identifiers including prefix or None.

Return type
list[str]

GetVolumeSnapshotIdentifiers(*volume_system*, *volume_identifiers*)

Retrieves volume snapshot identifiers.

This method can be used to prompt the user to provide volume snapshot identifiers.

Parameters

- **volume_system** (APFSVolumeSystem) – volume system.
- **volume_identifiers** (*list[str]*) – volume identifiers including prefix.

Returns
selected volume identifiers including prefix or None.

Return type
list[str]

abstract UnlockEncryptedVolume(*source_scanner_object*, *scan_context*, *locked_scan_node*, *credentials*)

Unlocks an encrypted volume.

This method can be used to prompt the user to provide encrypted volume credentials.

Parameters

- **source_scanner_object** (SourceScanner) – source scanner.
- **scan_context** (SourceScannerContext) – source scanner context.
- **locked_scan_node** (SourceScanNode) – locked scan node.
- **credentials** (Credentials) – credentials supported by the locked scan node.

Returns
True if the volume was unlocked.

Return type
bool

class dfvfs.helpers.volume_scanner.**VolumeScannerOptions**

Bases: object

Volume scanner options.

credentials

credentials, per type, to unlock volumes.

Type
list[tuple[str, str]]

partitions

partition identifiers.

Type
list[str]

scan_mode

mode that defines how the VolumeScanner should scan for volumes and snapshots.

Type
str

snapshots

snapshot identifiers.

Type
list[str]

volumes

volume identifiers, e.g. those of an APFS or LVM volume system.

Type
list[str]

SCAN_MODE_ALL = 'all'

SCAN_MODE_SNAPSHOTS_ONLY = 'snapshots-only'

SCAN_MODE_VOLUMES_ONLY = 'volumes-only'

class dfvfs.helpers.volume_scanner.**WindowsVolumeScanner**(*mediator=None*)

Bases: *VolumeScanner*

Windows volume scanner.

OpenFile(*windows_path*)

Opens the file specified by the Windows path.

Parameters
windows_path (*str*) – Windows path to the file.

Returns
file-like object or None if the file does not exist.

Return type
FileIO

ScanForWindowsVolume(*source_path*, *options=None*)

Scans for a Windows volume.

Parameters

- **source_path** (*str*) – source path.
- **options** (*Optional[VolumeScannerOptions]*) – volume scanner options. If None the default volume scanner options are used, which are defined in the VolumeScannerOptions class.

Returns
True if a Windows volume was found.

Return type
bool

Raises
ScannerError – if the source path does not exists, or if the source path is not a file or directory, or if the format of or within the source file is not supported.

dfvfs.helpers.windows_path_resolver module

A resolver for Windows paths to file system specific formats.

class dfvfs.helpers.windows_path_resolver.**WindowsPathResolver**(*file_system*, *mount_point*, *drive_letter='C'*)

Bases: `object`

Resolver object for Windows paths.

GetWindowsPath(*path_spec*)

Returns the Windows path based on a resolved path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns

corresponding Windows path or None if the Windows path could not
be determined.

Return type
str

Raises
PathSpecError – if the path specification is incorrect.

ResolvePath(*path*, *expand_variables=True*)

Resolves a Windows path in file system specific format.

Parameters

- **path** (*str*) – Windows path to resolve.
- **expand_variables** (*Optional[bool]*) – True if path variables should be expanded or not.

Returns
path specification in file system specific format.

Return type
PathSpec

SetEnvironmentVariable(*name*, *value*)

Sets an environment variable in the Windows path helper.

Parameters

- **name** (*str*) – name of the environment variable without enclosing %-characters, e.g. SystemRoot as in %SystemRoot%.
- **value** (*str*) – value of the environment variable.

Module contents

6.1.8 dfvfs.lib package

Submodules

dfvfs.lib.apfs_helper module

Helper functions for Apple File System (APFS) support.

dfvfs.lib.apfs_helper.**APFSContainerPathSpecGetVolumeIndex**(*path_spec*)

Retrieves the volume index from the path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
volume index or None if the index cannot be determined.

Return type
int

dfvfs.lib.apfs_helper.**APFSUnlockVolume**(*fsapfs_volume*, *path_spec*, *key_chain*)

Unlocks an APFS volume using the path specification.

Parameters

- **fsapfs_volume** (*pyapfs.volume*) – APFS volume.
- **path_spec** (PathSpec) – path specification.
- **key_chain** (KeyChain) – key chain.

Returns
True if the volume is unlocked, False otherwise.

Return type
bool

dfvfs.lib.bde_helper module

Helper function for BitLocker Drive Encryption (BDE) support.

dfvfs.lib.bde_helper.**BDEOpenVolume**(*bde_volume*, *path_spec*, *file_object*, *key_chain*)

Opens the BDE volume using the path specification.

Parameters

- **bde_volume** (*pybde.volume*) – BDE volume.
- **path_spec** (PathSpec) – path specification.
- **file_object** (FileIO) – file-like object.
- **key_chain** (KeyChain) – key chain.

dfvfs.lib.bde_helper.**BDEUnlockVolume**(*bde_volume*, *path_spec*, *key_chain*)

Unlocks a BDE volume using the path specification.

Parameters

- **bde_volume** (*pybde.volume*) – BDE volume.

- **path_spec** (PathSpec) – path specification.
- **key_chain** (KeyChain) – key chain.

Returns
True if the volume is unlocked, False otherwise.

Return type
bool

dfvfs.lib.cpio module

Copy in and out (CPIO) archive file.

class dfvfs.lib.cpio.**CPIOArchiveFile**(*encoding='utf-8'*)

Bases: *DataFormat*

CPIO archive file.

file_format

CPIO file format.

Type
str

Close()

Closes the CPIO archive file.

FileEntryExistsByPath(*path*)

Determines if file entry for a specific path exists.

Returns
True if the file entry exists.

Return type
bool

GetFileEntries(*path_prefix=''*)

Retrieves the file entries.

Parameters
path_prefix (*str*) – path prefix.

Yields
CPIOArchiveFileEntry – a CPIO archive file entry.

GetFileEntryByPath(*path*)

Retrieves a file entry for a specific path.

Returns
a CPIO archive file entry or None if not available.

Return type
CPIOArchiveFileEntry

Open(*file_object*)

Opens the CPIO archive file.

Parameters
file_object (FileIO) – a file-like object.

Raises

- **IOError** – if the file format signature is not supported.
- **OSError** – if the file format signature is not supported.

ReadDataAtOffset(*file_offset*, *size*)

Reads a byte string from the file-like object at a specific offset.

Parameters

- **file_offset** (*int*) – file offset.
- **size** (*int*) – number of bytes to read.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

property encoding

encoding of paths within the archive file.

Type
str

file_object = <_io.BufferedReader name='/home/docs/checkouts/readthedocs.org/user_builds/dfvfs/checkouts/latest/dfvfs/lib/cpio.yaml'>

class dfvfs.lib.cpio.**CPIOArchiveFileEntry**

Bases: object

CPIO archive file entry.

data_offset

data start offset.

Type
int

data_size

data size.

Type
int

group_identifier

group identifier (gid).

Type
int

inode_number

inode number.

Type
int

mode

mode.

Type
int

modification_time

modification POSIX timestamp.

Type
int

number_of_links

number of hard links.

Type
int

path

path.

Type
str

size

archive file entry record size.

Type
int

user_identifier

user identifier (uid).

Type
int

dfvfs.lib.cs_helper module

Helper function for Core Storage (CS) support.

dfvfs.lib.cs_helper.**CSPathSpecGetVolumeIndex**(*path_spec*)

Retrieves the volume index from the path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
volume index or None if not available.

Return type
int

dfvfs.lib.cs_helper.**CSUnlockLogicalVolume**(*fvde_logical_volume*, *path_spec*, *key_chain*)

Unlocks a Core Storage logical volume using the path specification.

Parameters

- **fvde_logical_volume** (*pyfvde.logical_volume*) – Core Storage logical volume.
- **path_spec** (PathSpec) – path specification.

- **key_chain** (KeyChain) – key chain.

Returns
True if the volume is unlocked, False otherwise.

Return type
bool

dfvfs.lib.data_format module

dtFabric data format helpers.

class dfvfs.lib.data_format.**DataFormat**

Bases: object

Data format.

dfvfs.lib.decorators module

Function decorators.

dfvfs.lib.decorators.**deprecated**(*function*)

Decorator to mark functions or methods as deprecated.

dfvfs.lib.definitions module

The Virtual File System (VFS) definitions.

dfvfs.lib.errors module

The Virtual File System (VFS) error classes.

exception dfvfs.lib.errors.**AccessError**

Bases: *Error*

Error indicating that a resource could not be accessed.

exception dfvfs.lib.errors.**BackEndError**

Bases: *Error*

Error indicating that a dependency has encountered a problem.

exception dfvfs.lib.errors.**CacheFullError**

Bases: *Error*

Error indicating a cache is full.

exception dfvfs.lib.errors.**Error**

Bases: Exception

Parent class for dfVFS specific errors.

exception dfvfs.lib.errors.**FileFormatError**

Bases: *Error*

Error indicating a problem in the format of a file.

exception dfvfs.lib.errors.**MountPointError**

Bases: *Error*

Error indicating a mount point does not exist.

exception dfvfs.lib.errors.**NotSupported**

Bases: *Error*

Error indicating that unsupported functionality was requested.

exception dfvfs.lib.errors.**PathSpecError**

Bases: *Error*

Error indicating a problem with a path specification.

exception dfvfs.lib.errors.**ScannerError**

Bases: *Error*

Error indicating that an item could not be scanned.

exception dfvfs.lib.errors.**UserAbort**

Bases: *Error*

Exception indicating that the user initiated an abort.

exception dfvfs.lib.errors.**VolumeSystemError**

Bases: *Error*

Error indicating a problem with a volume system.

dfvfs.lib.ewf_helper module

Helper functions for EWF image support.

dfvfs.lib.ewf_helper.**EWFGlobPathSpec**(*file_system*, *path_spec*)

Globs for path specifications according to the EWF naming schema.

Parameters

- **file_system** (FileSystem) – file system.
- **path_spec** (PathSpec) – path specification.

Returns

path specifications that match the glob.

Return type

list[*PathSpec*]

Raises

- ***PathSpecError*** – if the path specification is invalid.
- **RuntimeError** – if the maximum number of supported segment files is reached.

dfvfs.lib.glob2regex module

Glob to regular expression conversion.

Also see: https://en.wikipedia.org/wiki/Glob_(programming)

dfvfs.lib.glob2regex.**Glob2Regex**(*glob_pattern*)

Converts a glob pattern to a regular expression.

This function supports basic glob patterns that consist of: * matches everything ? matches any single character [seq] matches any character in sequence [!seq] matches any character not in sequence

Parameters
glob_pattern (*str*) – glob pattern.

Returns
regular expression pattern.

Return type
str

Raises
ValueError – if the glob pattern cannot be converted.

dfvfs.lib.gpt_helper module

Helper functions for GUID Partition Table (GPT) support.

dfvfs.lib.gpt_helper.**GPTPathSpecGetEntryIndex**(*path_spec*)

Retrieves the entry index from the path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
entry index or None if not available.

Return type
int

dfvfs.lib.gzipfile module

Gzip compressed stream file.

class dfvfs.lib.gzipfile.**GzipCompressedStream**

Bases: object

File-like object of a gzip compressed stream (file).

The gzip file format is defined in RFC1952: http://www.zlib.org/rfc-gzip.html

uncompressed_data_size

total size of the decompressed data stored in the gzip file.

Type
int

Open(*file_object*)

Opens the file-like object defined by path specification.

Parameters
file_object (FileIO) – file-like object that contains the gzip compressed stream.

Raises

- **IOError** – if the file-like object could not be opened.
- **OSError** – if the file-like object could not be opened.

close()

Closes the file-like object.

get_offset()

Retrieves the current offset into the file-like object.

Returns
current offset into the file-like object.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

get_size()

Retrieves the size of the file-like object.

Returns
size of the file-like object data.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

property members

members in the gzip file.

Type
list(*GzipMember*)

read(*size=None*)

Reads a byte string from the gzip file at the current offset.

The function will read a byte string up to the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file-like object.

Parameters

- **offset** (*int*) – offset to seek to.
- **whence** (*Optional[int]*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed or the file has not been opened.
- **OSError** – if the seek failed or the file has not been opened.

class dfvfs.lib.gzipfile.**GzipMember**(*file_object*, *member_start_offset*, *uncompressed_data_offset*)

Bases: *DataFormat*

Gzip member.

Gzip files have no index of members, so each member must be read sequentially before metadata and random seeks are possible. This class provides caching of gzip member data during the initial read of each member.

comment

comment stored in the member.

Type
str

member_end_offset

offset to the end of the member in the parent file object.

Type
int

member_start_offset

offset to the start of the member in the parent file object.

Type
int

operating_system

type of file system on which the compression took place.

Type
int

original_filename

original filename of the uncompressed file.

Type
str

uncompressed_data_offset

offset of the start of the uncompressed data in this member relative to the whole gzip file's uncompressed data.

Type
int

uncompressed_data_size
total size of the data in this gzip member after decompression.

Type
int

FlushCache()
Empties the cache that holds cached decompressed data.

ReadAtOffset(*offset*, *size=None*)
Reads a byte string from the gzip member at the specified offset.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters

- **offset** (*int*) – offset within the uncompressed data in this member to read from.
- **size** (*Optional[int]*) – maximum number of bytes to read, where None represents all remaining data, to a maximum of the uncompressed cache size.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **ValueError** – if a negative read size or offset is specified.

file_object = <_io.BufferedReader name='/home/docs/checkouts/readthedocs.org/user_builds/dfvfs/checkouts/latest/dfvfs/lib/gzipfile.yaml'>

dfvfs.lib.luksde_helper module

Helper function for LUKS Drive Encryption support.

dfvfs.lib.luksde_helper.**LUKSDEOpenVolume**(*luksde_volume*, *path_spec*, *file_object*, *key_chain*)
Opens the LUKSDE volume using the path specification.

Parameters

- **luksde_volume** (*pyluksde.volume*) – LUKSDE volume.
- **path_spec** (PathSpec) – path specification.
- **file_object** (FileIO) – file-like object.
- **key_chain** (KeyChain) – key chain.

dfvfs.lib.luksde_helper.**LUKSDEUnlockVolume**(*luksde_volume*, *path_spec*, *key_chain*)
Unlocks a LUKSDE volume using the path specification.

Parameters

- **luksde_volume** (*pyluksde.volume*) – LUKSDE volume.
- **path_spec** (PathSpec) – path specification.

- **key_chain** (KeyChain) – key chain.

Returns
True if the volume is unlocked, False otherwise.

Return type
bool

dfvfs.lib.lvm_helper module

Helper functions for Logical Volume Manager (LVM) support.

dfvfs.lib.lvm_helper.**LVMPathSpecGetVolumeIndex**(*path_spec*)

Retrieves the volume index from the path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
volume index or None if not available.

Return type
int

dfvfs.lib.raw_helper module

Helper functions for RAW storage media image support.

dfvfs.lib.raw_helper.**RawGlobPathSpec**(*file_system*, *path_spec*)

Globs for path specifications according to the split RAW naming schema.

Parameters

- **file_system** (FileSystem) – file system.
- **path_spec** (PathSpec) – path specification.

Returns
path specifications that match the glob.

Return type
list[*PathSpec*]

Raises

- ***PathSpecError*** – if the path specification is invalid.
- **RuntimeError** – if the maximum number of supported segment files is reached.

dfvfs.lib.sqlite_database module

Helper functions for SQLite database support.

class dfvfs.lib.sqlite_database.**SQLiteDatabaseFile**

Bases: object

SQLite database file using a file-like object.

`Close()`

Closes the database file object.

Raises

- **IOError** – if the close failed.
- **OSError** – if the close failed.

`GetNumberOfRows(table_name)`

Retrieves the number of rows in the table.

Parameters
table_name (*str*) – name of the table.

Returns
number of rows.

Return type
int

Raises

- **IOError** – if the file-like object has not been opened.
- **OSError** – if the file-like object has not been opened.

`HasColumn(table_name, column_name)`

Determines if a specific column exists.

Parameters

- **table_name** (*str*) – name of the table.
- **column_name** (*str*) – name of the column.

Returns
True if the column exists.

Return type
bool

Raises

- **IOError** – if the database file is not opened.
- **OSError** – if the database file is not opened.

`HasTable(table_name)`

Determines if a specific table exists.

Parameters
table_name (*str*) – name of the table.

Returns
True if the column exists.

Return type
bool

Raises

- **IOError** – if the database file is not opened.
- **OSError** – if the database file is not opened.

Open(*file_object*)

Opens the database file object.

Parameters
file_object (FileIO) – file-like object.

Raises

- **IOError** – if the SQLite database signature does not match.
- **OSError** – if the SQLite database signature does not match.
- **ValueError** – if the file-like object is invalid.

Query(*query*, *parameters=None*)

Queries the database file.

Parameters

- **query** (*str*) – SQL query.
- **parameters** (*Optional[dict|tuple]*) – query parameters.

Returns
rows resulting from the query.

Return type
list[sqlite3.Row]

dfvfs.lib.tsk_image module

Helper functions for SleuthKit (TSK) image support.

class dfvfs.lib.tsk_image.**TSKFileSystemImage**(**args: Any*, ***kwargs: Any*)

Bases: Img_Info

Pytsk3 image object using a file-like object.

close()

Closes the volume IO object.

get_size()

Retrieves the size.

read(*offset*, *size*)

Reads a byte string from the image object at the specified offset.

Parameters

- **offset** (*int*) – offset where to start reading.
- **size** (*int*) – number of bytes to read.

Returns
data read.

Return type
bytes

dfvfs.lib.tsk_partition module

Helper functions for SleuthKit (TSK) partition support.

dfvfs.lib.tsk_partition.**GetTSKVsPartByPathSpec**(*tsk_volume*, *path_spec*)

Retrieves the TSK volume system part object from the TSK volume object.

Parameters

- **tsk_volume** (*pytsk3.Volume_Info*) – TSK volume information.
- **path_spec** (PathSpec) – path specification.

Returns

containing:

pytsk3.TSK_VS_PART_INFO: TSK volume system part information or
None on error.

int: partition index or None if not available.

Return type
tuple

dfvfs.lib.tsk_partition.**TSKVolumeGetBytesPerSector**(*tsk_volume*)

Retrieves the number of bytes per sector from a TSK volume object.

Parameters
tsk_volume (*pytsk3.Volume_Info*) – TSK volume information.

Returns
number of bytes per sector or 512 by default.

Return type
int

dfvfs.lib.tsk_partition.**TSKVsPartGetNumberOfSectors**(*tsk_vs_part*)

Retrieves the number of sectors of a TSK volume system part object.

Parameters
tsk_vs_part (*pytsk3.TSK_VS_PART_INFO*) – TSK volume system part information.

Returns
number of sectors or None.

Return type
int

dfvfs.lib.tsk_partition.**TSKVsPartGetStartSector**(*tsk_vs_part*)

Retrieves the start sector of a TSK volume system part object.

Parameters
tsk_vs_part (*pytsk3.TSK_VS_PART_INFO*) – TSK volume system part information.

Returns
start sector or None.

Return type
int

dfvfs.lib.tsk_partition.**TSKVsPartIsAllocated**(*tsk_vs_part*)

Determines if the TSK volume system part object is allocated.

Parameters
tsk_vs_part (*pytsk3.TSK_VS_PART_INFO*) – TSK volume system part information.

Returns
True if the volume system part is allocated, False otherwise.

Return type
bool

dfvfs.lib.vshadow_helper module

Helper functions for Volume Shadow Snapshots (VSS) support.

dfvfs.lib.vshadow_helper.**VShadowPathSpecGetStoreIndex**(*path_spec*)

Retrieves the store index from the path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
store index or None if not available.

Return type
int

Module contents

6.1.9 dfvfs.mount package

Submodules

dfvfs.mount.manager module

The path specification mount point manager.

The mount point manager allows to "mount" one path specification onto another. This allows dfVFS to expose complex path specifications in a way closer to the original system interpretation.

E.g. the path specification: type=OS, location=/home/myuser/myimages/image.qcow2 type=QCOW type=TSK_PARTITION, location=/p1 type=TSK, inode=128, location=/Users/MyUser/MyFile.txt

could be mounted as: type=MOUNT, identifier=C type=TSK, inode=128, location=/Users/MyUser/MyFile.txt

where the "C" mount point would be: type=OS, location=/home/myuser/myimages/image.qcow2 type=QCOW type=TSK_PARTITION, location=/p1

class dfvfs.mount.manager.**MountPointManager**

Bases: object

Path specification mount point manager.

classmethod DeregisterMountPoint(*mount_point*)

Deregisters a path specification mount point.

Parameters
mount_point (*str*) – mount point identifier.

Raises
KeyError – if the corresponding mount point is not set.

classmethod GetMountPoint(*mount_point*)

Retrieves the path specification of a mount point.

Parameters
mount_point (*str*) – mount point identifier.

Returns

path specification of the mount point or None if the mount
point does not exists.

Return type
PathSpec

classmethod RegisterMountPoint(*mount_point*, *path_spec*)

Registers a path specification mount point.

Parameters

- **mount_point** (*str*) – mount point identifier.
- **path_spec** (PathSpec) – path specification of the mount point.

Raises
KeyError – if the corresponding mount point is already set.

Module contents

6.1.10 dfvfs.path package

Submodules

dfvfs.path.apfs_container_path_spec module

The APFS container path specification implementation.

class dfvfs.path.apfs_container_path_spec.**APFSContainerPathSpec**(*location=None*, *parent=None*, *volume_index=None*, ***kwargs*)

Bases: *PathSpec*

APFS container path specification implementation.

location

location.

Type
str

volume_index

volume index.

Type
int

TYPE_INDICATOR = 'APFS_CONTAINER'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.apfs_path_spec module

The APFS path specification implementation.

class dfvfs.path.apfs_path_spec.**APFSPathSpec**(*identifier=None*, *location=None*, *parent=None*, ***kwargs*)

Bases: *PathSpec*

APFS path specification implementation.

identifier

identifier.

Type
int

location

location.

Type
str

TYPE_INDICATOR = 'APFS'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.bde_path_spec module

The BitLocker Drive Encryption (BDE) path specification implementation.

class dfvfs.path.bde_path_spec.**BDEPathSpec**(*password=None*, *parent=None*, *recovery_password=None*, *startup_key=None*, ***kwargs*)

Bases: *PathSpec*

BDE path specification.

password

password.

Type
str

recovery_password

recovery password.

Type
str

startup_key

name of the startup key file.

Type
str

TYPE_INDICATOR = 'BDE'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.compressed_stream_path_spec module

The compressed stream path specification implementation.

class dfvfs.path.compressed_stream_path_spec.**CompressedStreamPathSpec**(*compression_method=None*, *parent=None*, ***kwargs*)

Bases: *PathSpec*

Compressed stream path specification.

compression_method

method used to the compress the data.

Type
str

TYPE_INDICATOR = 'COMPRESSED_STREAM'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.cpio_path_spec module

The CPIO path specification implementation.

class dfvfs.path.cpio_path_spec.**CPIOPathSpec**(*location=None*, *parent=None*, ***kwargs*)

Bases: *LocationPathSpec*

CPIO file path specification.

TYPE_INDICATOR = 'CPIO'

dfvfs.path.cs_path_spec module

The Core Storage (CS) path specification implementation.

class dfvfs.path.cs_path_spec.**CSPathSpec**(*encrypted_root_plist=None*, *location=None*, *password=None*, *parent=None*, *recovery_password=None*, *volume_index=None*, ***kwargs*)

Bases: *PathSpec*

CS path specification.

encrypted_root_plist

path to the EncryptedRoot.plist.wipekey file.

Type
str

location

location.

Type
str

password

password.

Type
str

recovery_password

recovery password.

Type
str

volume_index

logical volume index.

Type
int

TYPE_INDICATOR = 'CS'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.data_range_path_spec module

The data range path specification implementation.

class dfvfs.path.data_range_path_spec.**DataRangePathSpec**(*parent=None*, *range_offset=None*, *range_size=None*, ***kwargs*)

Bases: *PathSpec*

Data range path specification.

range_offset

start offset of the data range.

Type
int

range_size

size of the data range.

Type
int

TYPE_INDICATOR = 'DATA_RANGE'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.encoded_stream_path_spec module

The encoded stream path specification implementation.

class dfvfs.path.encoded_stream_path_spec.**EncodedStreamPathSpec**(*encoding_method=None*, *parent=None*, ***kwargs*)

Bases: *PathSpec*

Encoded stream path specification.

encoding_method

method used to the encode the data.

Type
str

TYPE_INDICATOR = 'ENCODED_STREAM'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.encrypted_stream_path_spec module

The encrypted stream path specification implementation.

class dfvfs.path.encrypted_stream_path_spec.**EncryptedStreamPathSpec**(*cipher_mode=None*, *encryption_method=None*, *initialization_vector=None*, *key=None*, *parent=None*, ***kwargs*)

Bases: *PathSpec*

Encrypted stream path specification.

cipher_mode

cipher mode.

Type
str

encryption_method

method used to the encrypt the data.

Type
str

initialization_vector

initialization vector.

Type
bytes

key

key.

Type
bytes

TYPE_INDICATOR = 'ENCRYPTED_STREAM'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.ewf_path_spec module

The EWF image path specification implementation.

class dfvfs.path.ewf_path_spec.**EWFPathSpec**(*parent=None*, ***kwargs*)

Bases: *PathSpec*

EWF image path specification.

TYPE_INDICATOR = 'EWF'

dfvfs.path.ext_path_spec module

The EXT path specification implementation.

class dfvfs.path.ext_path_spec.**EXTPathSpec**(*inode=None*, *location=None*, *parent=None*, ***kwargs*)

Bases: *PathSpec*

EXT path specification implementation.

inode

inode.

Type
int

location

location.

Type
str

TYPE_INDICATOR = 'EXT'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.factory module

The path specification factory.

class dfvfs.path.factory.**Factory**

Bases: object

Path specification factory.

classmethod DeregisterPathSpec(*path_spec_type*)

Deregisters a path specification type.

Parameters
path_spec_type (*type*) – path specification type.

Raises
KeyError – if path specification type is not registered.

classmethod GetProperties(*path_spec*)

Retrieves a dictionary containing the path specification properties.

Parameters
path_spec (PathSpec) – path specification.

Returns
path specification properties.

Return type
dict[str, str]

Raises
dict – path specification properties.

classmethod IsSystemLevelTypeIndicator(*type_indicator*)

Determines if the type indicator is at system-level.

Parameters
type_indicator (*str*) – type indicator.

Returns
True if the type indicator is at system-level.

Return type
bool

classmethod NewPathSpec(*type_indicator*, ***kwargs*)

Creates a new path specification for the specific type indicator.

Parameters

- **type_indicator** (*str*) – type indicator.
- **kwargs** (*dict*) – keyword arguments depending on the path specification.

Returns
path specification.

Return type
PathSpec

Raises
KeyError – if path specification is not registered.

PROPERTY_NAMES = frozenset({'cipher_mode', 'column_name', 'compression_method', 'data_stream', 'encoding_method', 'encryption_method', 'identifier', 'initialization_vector', 'inode', 'key', 'location', 'mft_attribute', 'mft_entry', 'part_index', 'password', 'range_offset', 'range_size', 'recovery_password', 'row_condition', 'row_index', 'start_offset', 'startup_key', 'store_index', 'table_name', 'volume_index'})

classmethod RegisterPathSpec(*path_spec_type*)

Registers a path specification type.

Parameters
path_spec_type (*type*) – path specification type.

Raises
KeyError – if path specification type is already registered.

dfvfs.path.fake_path_spec module

The fake path specification implementation.

class dfvfs.path.fake_path_spec.**FakePathSpec**(*location=None*, ***kwargs*)

Bases: *LocationPathSpec*

Fake path specification.

TYPE_INDICATOR = 'FAKE'

dfvfs.path.fat_path_spec module

The FAT path specification implementation.

class dfvfs.path.fat_path_spec.**FATPathSpec**(*identifier=None*, *location=None*, *parent=None*, ***kwargs*)

Bases: *PathSpec*

FAT path specification implementation.

identifier

(virtual) identifier.

Type
int

location

location.

Type
str

TYPE_INDICATOR = 'FAT'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.gpt_path_spec module

The GUID Partition Table (GPT) path specification implementation.

class dfvfs.path.gpt_path_spec.**GPTPathSpec**(*location=None*, *entry_index=None*, *parent=None*, ***kwargs*)

Bases: *PathSpec*

GPT path specification.

entry_index

partition entry index.

Type
int

location

location.

Type
str

TYPE_INDICATOR = 'GPT'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.gzip_path_spec module

The gzip file path specification implementation.

class dfvfs.path.gzip_path_spec.**GzipPathSpec**(*parent=None*, ***kwargs*)

Bases: *PathSpec*

Gzip file path specification.

TYPE_INDICATOR = 'GZIP'

dfvfs.path.hfs_path_spec module

The HFS path specification implementation.

class dfvfs.path.hfs_path_spec.**HFSPathSpec**(*data_stream=None*, *identifier=None*, *location=None*, *parent=None*, ***kwargs*)

Bases: *PathSpec*

HFS path specification implementation.

data_stream

data stream name, where None indicates the default data stream.

Type
str

identifier

catalog node identifier (CNID).

Type
int

location

location.

Type
str

TYPE_INDICATOR = 'HFS'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.location_path_spec module

The location-based path specification implementation.

class dfvfs.path.location_path_spec.**LocationPathSpec**(*location=None*, *parent=None*, ***kwargs*)

Bases: *PathSpec*

Base class for location-based path specifications.

location

location.

Type
str

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.luksde_path_spec module

The LUKS Drive Encryption path specification implementation.

class dfvfs.path.luksde_path_spec.**LUKSDEPathSpec**(*password=None*, *parent=None*, ***kwargs*)

Bases: *PathSpec*

LUKSDE path specification.

password

password.

Type
str

TYPE_INDICATOR = 'LUKSDE'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.lvm_path_spec module

The Logical Volume Manager (LVM) path specification implementation.

class dfvfs.path.lvm_path_spec.**LVMPathSpec**(*location=None*, *parent=None*, *volume_index=None*, ***kwargs*)

Bases: *PathSpec*

LVM path specification.

location

location.

Type
str

volume_index

logical volume index.

Type
int

TYPE_INDICATOR = 'LVM'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.modi_path_spec module

The Mac OS disk image path specification implementation.

class dfvfs.path.modi_path_spec.**MODIPathSpec**(*parent=None*, ***kwargs*)

Bases: *PathSpec*

Mac OS disk image path specification.

TYPE_INDICATOR = 'MODI'

dfvfs.path.mount_path_spec module

The mount path specification implementation.

class dfvfs.path.mount_path_spec.**MountPathSpec**(*identifier=None*, ***kwargs*)

Bases: *PathSpec*

Mount path specification.

identifier

identifier of the mount point.

Type
str

TYPE_INDICATOR = 'MOUNT'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.ntfs_path_spec module

The path NTFS specification implementation.

class dfvfs.path.ntfs_path_spec.**NTFSPathSpec**(*data_stream=None*, *location=None*, *mft_attribute=None*, *mft_entry=None*, *parent=None*, ***kwargs*)

Bases: *PathSpec*

NTFS path specification.

data_stream

data stream name, where None indicates the default data stream.

Type
str

location

location.

Type
str

mft_attribute

$FILE_NAME MFT attribute index, where the first attribute is indicated by 0.

Type
int

mft_entry

MFT entry, where the first entry is indicated by 0.

Type
int

TYPE_INDICATOR = 'NTFS'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.os_path_spec module

The operating system path specification implementation.

class dfvfs.path.os_path_spec.**OSPathSpec**(*location=None*, ***kwargs*)

Bases: *LocationPathSpec*

Operating system path specification.

TYPE_INDICATOR = 'OS'

dfvfs.path.path_spec module

The Virtual File System (VFS) path specification interface.

class dfvfs.path.path_spec.**PathSpec**(*parent=None*, ***kwargs*)

Bases: object

Path specification interface.

parent

parent path specification.

Type
PathSpec

CopyToDict()

Copies the path specification to a dictionary.

Returns
path specification attributes.

Return type
dict[str, object]

`HasParent()`

Determines if the path specification has a parent.

Returns
True if the path specification has a parent.

Return type
bool

`IsFileSystem()`

Determines if the path specification is a file system.

Returns
True if the path specification is a file system.

Return type
bool

`IsSystemLevel()`

Determines if the path specification is at system-level.

System-level is an indication used if the path specification is handled by the operating system and should not have a parent.

Returns
True if the path specification is at system-level.

Return type
bool

`IsVolumeSystem()`

Determines if the path specification is a volume system.

Returns
True if the path specification is a volume system.

Return type
bool

`IsVolumeSystemRoot()`

Determines if the path specification is the root of a volume system.

Returns
True if the path specification is the root of a volume system.

Return type
bool

`__eq__(`*other*`)`

Determines if the path specification is equal to the other.

`__hash__()`

Returns the hash of a path specification.

`property comparable`

comparable representation of the path specification.

Type
str

property type_indicator
type indicator.

Type
str

dfvfs.path.phdi_path_spec module

The PHDI image path specification implementation.

class dfvfs.path.phdi_path_spec.**PHDIPathSpec**(*parent=None*, ***kwargs*)
Bases: *PathSpec*

PHDI image path specification.

TYPE_INDICATOR = 'PHDI'

dfvfs.path.qcow_path_spec module

The QCOW image path specification implementation.

class dfvfs.path.qcow_path_spec.**QCOWPathSpec**(*parent=None*, ***kwargs*)
Bases: *PathSpec*

QCOW image path specification.

TYPE_INDICATOR = 'QCOW'

dfvfs.path.raw_path_spec module

The RAW storage media image path specification implementation.

class dfvfs.path.raw_path_spec.**RawPathSpec**(*parent=None*, ***kwargs*)
Bases: *PathSpec*

RAW storage media image path specification.

TYPE_INDICATOR = 'RAW'

dfvfs.path.sqlite_blob_path_spec module

The SQLite blob path specification implementation.

class dfvfs.path.sqlite_blob_path_spec.**SQLiteBlobPathSpec**(*column_name=None*, *parent=None*, *row_condition=None*, *row_index=None*, *table_name=None*, ***kwargs*)
Bases: *PathSpec*

SQLite blob file path specification.

column_name
name of the column in which the blob is stored.

Type
str

row_condition

condition of the row in which the blob is stored. The condition is a tuple in the form: (column_name, operator, value). The condition must yield a single result.

Type
tuple

row_index

index of the row in which the blob is stored.

Type
int

table_name

name of the table in which the blob is stored.

Type
str

TYPE_INDICATOR = 'SQLITE_BLOB'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.tar_path_spec module

The TAR path specification implementation.

class dfvfs.path.tar_path_spec.**TARPathSpec**(*location=None*, *parent=None*, ***kwargs*)

Bases: *LocationPathSpec*

TAR file path specification.

TYPE_INDICATOR = 'TAR'

dfvfs.path.tsk_partition_path_spec module

The SleuthKit (TSK) partition path specification implementation.

class dfvfs.path.tsk_partition_path_spec.**TSKPartitionPathSpec**(*location=None*, *parent=None*, *part_index=None*, *start_offset=None*, ***kwargs*)

Bases: *PathSpec*

SleuthKit (TSK) partition path specification.

location

location.

Type
str

part_index

part index.

Type
int

start_offset

start offset.

Type
int

TYPE_INDICATOR = 'TSK_PARTITION'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.tsk_path_spec module

The SleuthKit (TSK) path specification implementation.

class dfvfs.path.tsk_path_spec.**TSKPathSpec**(*data_stream=None*, *inode=None*, *location=None*, *parent=None*, ***kwargs*)

Bases: *PathSpec*

SleuthKit (TSK) path specification.

data_stream

data stream name, where None indicates the default data stream.

Type
str

inode

inode.

Type
int

location

location.

Type
str

TYPE_INDICATOR = 'TSK'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.vhdi_path_spec module

The Virtual Hard Disk image path specification implementation.

class dfvfs.path.vhdi_path_spec.**VHDIPathSpec**(*parent=None*, ***kwargs*)

Bases: *PathSpec*

Virtual Hard Disk image path specification.

TYPE_INDICATOR = 'VHDI'

dfvfs.path.vmdk_path_spec module

The VMDK image path specification implementation.

class dfvfs.path.vmdk_path_spec.**VMDKPathSpec**(*parent=None*, ***kwargs*)

Bases: *PathSpec*

VMDK image path specification.

TYPE_INDICATOR = 'VMDK'

dfvfs.path.vshadow_path_spec module

The Volume Shadow Snapshots (VSS) path specification implementation.

class dfvfs.path.vshadow_path_spec.**VShadowPathSpec**(*location=None*, *parent=None*, *store_index=None*, ***kwargs*)

Bases: *PathSpec*

Volume Shadow Snapshots (VSS) path specification.

location

location.

Type
str

store_index

store index.

Type
int

TYPE_INDICATOR = 'VSHADOW'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.xfs_path_spec module

The XFS path specification implementation.

class dfvfs.path.xfs_path_spec.**XFSPathSpec**(*inode=None*, *location=None*, *parent=None*, ***kwargs*)

Bases: *PathSpec*

XFS path specification implementation.

inode

inode.

Type
int

location

location.

Type
str

TYPE_INDICATOR = 'XFS'

property comparable

comparable representation of the path specification.

Type
str

dfvfs.path.zip_path_spec module

The ZIP archive file path specification implementation.

class dfvfs.path.zip_path_spec.**ZipPathSpec**(*location=None*, *parent=None*, ***kwargs*)

Bases: *LocationPathSpec*

ZIP archive file path specification.

TYPE_INDICATOR = 'ZIP'

Module contents

Imports for path specification factory.

6.1.11 dfvfs.resolver package

Submodules

dfvfs.resolver.context module

The resolver context object.

class dfvfs.resolver.context.**Context**

Bases: object

Resolver context.

CacheFileObject(*path_spec*, *file_object*)

Caches a file-like object based on a path specification.

Parameters

- **path_spec** (PathSpec) – path specification.
- **file_object** (FileIO) – file-like object.

Raises

KeyError – if the file object already is cached.

CacheFileSystem(*path_spec*, *file_system*)

Caches a file system object based on a path specification.

Parameters

- **path_spec** (PathSpec) – path specification.
- **file_system** (FileSystem) – file system object.

Raises

KeyError – if the file system already is cached.

DeregisterMountPoint(*mount_point*)

Deregisters a path specification mount point.

Parameters

mount_point (*str*) – mount point identifier.

Raises

KeyError – if the corresponding mount point is not set.

Empty()

Empties the caches.

GetFileObject(*path_spec*)

Retrieves a file-like object defined by path specification.

Parameters

path_spec (PathSpec) – path specification.

Returns

a file-like object or None if not cached.

Return type

FileIO

GetFileSystem(*path_spec*)

Retrieves a file system object defined by path specification.

Parameters

path_spec (PathSpec) – path specification.

Returns

a file system object or None if not cached.

Return type

FileSystem

GetMountPoint(*mount_point*)

Retrieves the path specification of a mount point.

Parameters
mount_point (*str*) – mount point identifier.

Returns
path specification of the mount point or None if the mount point does not exists.

Return type
PathSpec

RegisterMountPoint(*mount_point*, *path_spec*)

Registers a path specification mount point.

Parameters
- **mount_point** (*str*) – mount point identifier.
- **path_spec** (PathSpec) – path specification of the mount point.

Raises
KeyError – if the corresponding mount point is already set.

dfvfs.resolver.resolver module

The path specification resolver.

class dfvfs.resolver.resolver.**Resolver**

Bases: object

Path specification resolver.

classmethod OpenFileEntry(*path_spec_object*, *resolver_context=None*)

Opens a file entry object defined by path specification.

Parameters
- **path_spec_object** (PathSpec) – path specification.
- **resolver_context** (*Optional[Context]*) – resolver context, where None represents the built in context which is not multi process safe.

Returns
file entry or None if the path specification could not be resolved.

Return type
FileEntry

classmethod OpenFileObject(*path_spec_object*, *resolver_context=None*)

Opens a file-like object defined by path specification.

Parameters
- **path_spec_object** (PathSpec) – path specification.
- **resolver_context** (*Optional[Context]*) – resolver context, where None represents the built in context which is not multi process safe.

Returns
file-like object or None if the path specification could not be resolved.

Return type
FileIO

Raises

- ***BackEndError*** – if the file object cannot be opened.
- ***MountPointError*** – if the mount point specified in the path specification does not exist.
- ***PathSpecError*** – if the path specification is incorrect.
- **TypeError** – if the path specification type is unsupported.

classmethod OpenFileSystem(*path_spec_object*, *resolver_context=None*)

Opens a file system object defined by path specification.

Parameters

- **path_spec_object** (PathSpec) – path specification.
- **resolver_context** (*Optional[Context]*) – resolver context, where None represents the built in context which is not multi process safe.

Returns

file system or None if the path specification could not
be resolved or has no file system object.

Return type
FileSystem

Raises

- ***AccessError*** – if the access to open the file system was denied.
- ***BackEndError*** – if the file system cannot be opened.
- ***MountPointError*** – if the mount point specified in the path specification does not exist.
- ***PathSpecError*** – if the path specification is incorrect.
- **TypeError** – if the path specification type is unsupported.

key_chain = <dfvfs.credentials.keychain.KeyChain object>

Module contents

6.1.12 dfvfs.resolver_helpers package

Submodules

dfvfs.resolver_helpers.apfs_container_resolver_helper module

The APFS container path specification resolver helper implementation.

class dfvfs.resolver_helpers.apfs_container_resolver_helper.**APFSContainerResolverHelper**

Bases: *ResolverHelper*

APFS container resolver helper.

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
APFSContainerFileSystem

TYPE_INDICATOR = 'APFS_CONTAINER'

dfvfs.resolver_helpers.apfs_resolver_helper module

The APFS path specification resolver helper implementation.

class dfvfs.resolver_helpers.apfs_resolver_helper.**APFSResolverHelper**

Bases: *ResolverHelper*

APFS resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

TYPE_INDICATOR = 'APFS'

dfvfs.resolver_helpers.bde_resolver_helper module

The BDE volume path specification resolver helper implementation.

class dfvfs.resolver_helpers.bde_resolver_helper.**BDEResolverHelper**

Bases: *ResolverHelper*

BDE volume resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
BDEFile

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
BDEFileSystem

TYPE_INDICATOR = 'BDE'

dfvfs.resolver_helpers.compressed_stream_resolver_helper module

The compressed stream path specification resolver helper implementation.

class
dfvfs.resolver_helpers.compressed_stream_resolver_helper.**CompressedStreamResolverHelper**

Bases: *ResolverHelper*

Compressed stream resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

TYPE_INDICATOR = 'COMPRESSED_STREAM'

dfvfs.resolver_helpers.cpio_resolver_helper module

The CPIO path specification resolver helper implementation.

class dfvfs.resolver_helpers.cpio_resolver_helper.**CPIOResolverHelper**

Bases: *ResolverHelper*

CPIO resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

TYPE_INDICATOR = 'CPIO'

dfvfs.resolver_helpers.cs_resolver_helper module

The CS path specification resolver helper implementation.

class dfvfs.resolver_helpers.cs_resolver_helper.**CSResolverHelper**

Bases: *ResolverHelper*

Logical Volume Manager (CS) resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

TYPE_INDICATOR = 'CS'

dfvfs.resolver_helpers.data_range_resolver_helper module

The data range path specification resolver helper implementation.

class dfvfs.resolver_helpers.data_range_resolver_helper.**DataRangeResolverHelper**

Bases: *ResolverHelper*

Data range resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

TYPE_INDICATOR = 'DATA_RANGE'

dfvfs.resolver_helpers.encoded_stream_resolver_helper module

The encoded stream path specification resolver helper implementation.

class dfvfs.resolver_helpers.encoded_stream_resolver_helper.**EncodedStreamResolverHelper**

Bases: *ResolverHelper*

Encoded stream resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

TYPE_INDICATOR = 'ENCODED_STREAM'

dfvfs.resolver_helpers.encrypted_stream_resolver_helper module

The encrypted stream path specification resolver helper implementation.

class dfvfs.resolver_helpers.encrypted_stream_resolver_helper.**EncryptedStreamResolverHelper**

Bases: *ResolverHelper*

Encrypted stream resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

TYPE_INDICATOR = 'ENCRYPTED_STREAM'

dfvfs.resolver_helpers.ewf_resolver_helper module

The EWF image path specification resolver helper implementation.

class dfvfs.resolver_helpers.ewf_resolver_helper.**EWFResolverHelper**

Bases: *ResolverHelper*

EWF image resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

`TYPE_INDICATOR = 'EWF'`

dfvfs.resolver_helpers.ext_resolver_helper module

The EXT path specification resolver helper implementation.

class dfvfs.resolver_helpers.ext_resolver_helper.**EXTResolverHelper**

Bases: *ResolverHelper*

EXT resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

`TYPE_INDICATOR = 'EXT'`

dfvfs.resolver_helpers.fake_resolver_helper module

The fake path specification resolver helper implementation.

class dfvfs.resolver_helpers.fake_resolver_helper.**FakeResolverHelper**

Bases: *ResolverHelper*

Fake resolver helper.

`TYPE_INDICATOR = 'FAKE'`

dfvfs.resolver_helpers.fat_resolver_helper module

The FAT path specification resolver helper implementation.

class dfvfs.resolver_helpers.fat_resolver_helper.**FATResolverHelper**

Bases: *ResolverHelper*

FAT resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

TYPE_INDICATOR = 'FAT'

dfvfs.resolver_helpers.gpt_resolver_helper module

The GPT path specification resolver helper implementation.

class dfvfs.resolver_helpers.gpt_resolver_helper.**GPTResolverHelper**

Bases: *ResolverHelper*

GUID Partition Table (GPT) resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

TYPE_INDICATOR = 'GPT'

dfvfs.resolver_helpers.gzip_resolver_helper module

The gzip file path specification resolver helper implementation.

class dfvfs.resolver_helpers.gzip_resolver_helper.**GzipResolverHelper**

Bases: *ResolverHelper*

Gzip file resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

TYPE_INDICATOR = 'GZIP'

dfvfs.resolver_helpers.hfs_resolver_helper module

The HFS path specification resolver helper implementation.

class dfvfs.resolver_helpers.hfs_resolver_helper.**HFSResolverHelper**

Bases: *ResolverHelper*

HFS resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

TYPE_INDICATOR = 'HFS'

dfvfs.resolver_helpers.luksde_resolver_helper module

The LUKSDE volume path specification resolver helper implementation.

class dfvfs.resolver_helpers.luksde_resolver_helper.**LUKSDEResolverHelper**

Bases: *ResolverHelper*

LUKSDE volume resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
LUKSDEFile

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
LUKSDEFileSystem

TYPE_INDICATOR = 'LUKSDE'

dfvfs.resolver_helpers.lvm_resolver_helper module

The LVM path specification resolver helper implementation.

class dfvfs.resolver_helpers.lvm_resolver_helper.**LVMResolverHelper**

Bases: *ResolverHelper*

Logical Volume Manager (LVM) resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

TYPE_INDICATOR = 'LVM'

dfvfs.resolver_helpers.manager module

The path specification resolver helper manager.

class dfvfs.resolver_helpers.manager.**ResolverHelperManager**

Bases: object

Path specification resolver helper manager.

classmethod DeregisterHelper(*resolver_helper*)

Deregisters a path specification resolver helper.

Parameters
resolver_helper (ResolverHelper) – resolver helper.

Raises
KeyError – if resolver helper object is not set for the corresponding type indicator.

classmethod GetHelper(*type_indicator*)

Retrieves the path specification resolver helper for the specified type.

Parameters
type_indicator (*str*) – type indicator.

Returns
a resolver helper.

Return type
ResolverHelper

Raises
KeyError – if resolver helper is not set for the corresponding type indicator.

classmethod RegisterHelper(*resolver_helper*)

Registers a path specification resolver helper.

Parameters
resolver_helper (ResolverHelper) – resolver helper.

Raises
KeyError – if resolver helper object is already set for the corresponding type indicator.

dfvfs.resolver_helpers.modi_resolver_helper module

The Mac OS disk image path specification resolver helper implementation.

class dfvfs.resolver_helpers.modi_resolver_helper.**MODIResolverHelper**

Bases: *ResolverHelper*

Mac OS disk image resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

TYPE_INDICATOR = 'MODI'

dfvfs.resolver_helpers.ntfs_resolver_helper module

The NTFS path specification resolver helper implementation.

class dfvfs.resolver_helpers.ntfs_resolver_helper.**NTFSResolverHelper**

Bases: *ResolverHelper*

NTFS resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

TYPE_INDICATOR = 'NTFS'

dfvfs.resolver_helpers.os_resolver_helper module

The operating system path specification resolver helper implementation.

class dfvfs.resolver_helpers.os_resolver_helper.**OSResolverHelper**

Bases: *ResolverHelper*

Operating system resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.

- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

TYPE_INDICATOR = 'OS'

dfvfs.resolver_helpers.phdi_resolver_helper module

The PHDI image path specification resolver helper implementation.

class dfvfs.resolver_helpers.phdi_resolver_helper.**PHDIResolverHelper**

Bases: *ResolverHelper*

PHDI image resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

TYPE_INDICATOR = 'PHDI'

dfvfs.resolver_helpers.qcow_resolver_helper module

The QCOW image path specification resolver helper implementation.

class dfvfs.resolver_helpers.qcow_resolver_helper.**QCOWResolverHelper**

Bases: *ResolverHelper*

QCOW image resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

TYPE_INDICATOR = 'QCOW'

dfvfs.resolver_helpers.raw_resolver_helper module

The RAW image path specification resolver helper implementation.

class dfvfs.resolver_helpers.raw_resolver_helper.**RawResolverHelper**

Bases: *ResolverHelper*

RAW storage media image resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

TYPE_INDICATOR = 'RAW'

dfvfs.resolver_helpers.resolver_helper module

The Virtual File System (VFS) resolver helper interface.

class dfvfs.resolver_helpers.resolver_helper.**ResolverHelper**

Bases: object

Resolver helper interface.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

Raises
NotSupported – if there is no implementation to create a file input/output (IO) object.

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

Raises
NotSupported – if there is no implementation to create a file system.

property type_indicator

type indicator.

Type
str

dfvfs.resolver_helpers.sqlite_blob_resolver_helper module

The SQLite blob file path specification resolver helper implementation.

class dfvfs.resolver_helpers.sqlite_blob_resolver_helper.**SQLiteBlobResolverHelper**

Bases: *ResolverHelper*

SQLite blob resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
SQLiteBlobFile

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
SQLiteBlobFileSystem

TYPE_INDICATOR = 'SQLITE_BLOB'

dfvfs.resolver_helpers.tar_resolver_helper module

The TAR path specification resolver helper implementation.

class dfvfs.resolver_helpers.tar_resolver_helper.**TARResolverHelper**

Bases: *ResolverHelper*

TAR resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

`TYPE_INDICATOR = 'TAR'`

dfvfs.resolver_helpers.tsk_partition_resolver_helper module

The TSK partition path specification resolver helper implementation.

`class dfvfs.resolver_helpers.tsk_partition_resolver_helper.TSKPartitionResolverHelper`

Bases: *ResolverHelper*

SleuthKit (TSK) partition presolver helper.

`NewFileObject(resolver_context, path_spec)`

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

`NewFileSystem(resolver_context, path_spec)`

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

`TYPE_INDICATOR = 'TSK_PARTITION'`

dfvfs.resolver_helpers.tsk_resolver_helper module

The SleuthKit (TSK) path specification resolver helper implementation.

`class dfvfs.resolver_helpers.tsk_resolver_helper.TSKResolverHelper`

Bases: *ResolverHelper*

SleuthKit (TSK) resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns

file input/output (IO) object.

Return type

FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns

file system.

Return type

FileSystem

TYPE_INDICATOR = 'TSK'

dfvfs.resolver_helpers.vhdi_resolver_helper module

The VHD image path specification resolver helper implementation.

class dfvfs.resolver_helpers.vhdi_resolver_helper.**VHDIResolverHelper**

Bases: *ResolverHelper*

VHD image resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns

file input/output (IO) object.

Return type

FileIO

TYPE_INDICATOR = 'VHDI'

dfvfs.resolver_helpers.vmdk_resolver_helper module

The VMDK image path specification resolver helper implementation.

class dfvfs.resolver_helpers.vmdk_resolver_helper.**VMDKResolverHelper**

Bases: *ResolverHelper*

VMDK image resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

TYPE_INDICATOR = 'VMDK'

dfvfs.resolver_helpers.vshadow_resolver_helper module

The VSS path specification resolver helper implementation.

class dfvfs.resolver_helpers.vshadow_resolver_helper.**VShadowResolverHelper**

Bases: *ResolverHelper*

Volume Shadow Snapshots (VSS) resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

`TYPE_INDICATOR = 'VSHADOW'`

dfvfs.resolver_helpers.xfs_resolver_helper module

The XFS path specification resolver helper implementation.

class dfvfs.resolver_helpers.xfs_resolver_helper.**XFSResolverHelper**

Bases: *ResolverHelper*

XFS resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

`TYPE_INDICATOR = 'XFS'`

dfvfs.resolver_helpers.zip_resolver_helper module

The ZIP path specification resolver helper implementation.

class dfvfs.resolver_helpers.zip_resolver_helper.**ZipResolverHelper**

Bases: *ResolverHelper*

ZIP resolver helper.

NewFileObject(*resolver_context*, *path_spec*)

Creates a new file input/output (IO) object.

Parameters

- **resolver_context** (Context) – resolver context.

- **path_spec** (PathSpec) – a path specification.

Returns
file input/output (IO) object.

Return type
FileIO

NewFileSystem(*resolver_context*, *path_spec*)

Creates a new file system object.

Parameters

- **resolver_context** (Context) – resolver context.
- **path_spec** (PathSpec) – a path specification.

Returns
file system.

Return type
FileSystem

TYPE_INDICATOR = 'ZIP'

Module contents

Imports for the path specification resolver.

6.1.13 dfvfs.serializer package

Submodules

dfvfs.serializer.json_serializer module

The JSON serializer object implementation.

class dfvfs.serializer.json_serializer.**JsonPathSpecSerializer**

Bases: *PathSpecSerializer*

JSON path specification serializer object.

classmethod ReadSerialized(*json_string*)

Reads a path specification from serialized form.

Parameters
json_string (*str*) – JSON serialized path specification.

Returns
a path specification.

Return type
PathSpec

classmethod WriteSerialized(*path_spec_object*)

Writes a path specification to serialized form.

Parameters
path_spec_object (PathSpec) – a path specification.

Returns
JSON serialized path specification.

Return type
str

dfvfs.serializer.serializer module

The Virtual File System (VFS) serializer object interfaces.

class dfvfs.serializer.serializer.**PathSpecSerializer**

Bases: object

Path specification serializer interface.

abstract ReadSerialized(*serialized*)

Reads a path specification from serialized form.

Parameters
serialized (*object*) – serialized form of the path specification.

Returns
a path specification.

Return type
PathSpec

abstract WriteSerialized(*path_spec*)

Writes a path specification to serialized form.

Parameters
path_spec (PathSpec) – a path specification.

Returns
serialized form of the path specification.

Return type
object

Module contents

6.1.14 dfvfs.vfs package

Submodules

dfvfs.vfs.apfs_attribute module

The APFS attribute implementation.

class dfvfs.vfs.apfs_attribute.**APFSExtendedAttribute**(*fsapfs_extended_attribute*)

Bases: *Attribute*

APFS extended attribute that uses pyfsapfs.

GetExtents()

Retrieves the extents.

Returns
the extents of the attribute data.

Return type
list[*Extent*]

`get_offset()`

Retrieves the current offset into the file input/output (IO) object.

Returns
current offset into the file input/output (IO) object.

Return type
int

`get_size()`

Retrieves the size of the file input/output (IO) object.

Returns
size of the file input/output (IO) object.

Return type
int

`property name`

name.

Type
str

`read(size=None)`

Reads a byte string from the file input/output (IO) object.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

`seek(offset, whence=0)`

Seeks to an offset within the file input/output (IO) object.

Parameters

- **offset** (*int*) – offset to seek.
- **whence** (*Optional[int]*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

seekable()

Determines if a file input/output (IO) object is seekable.

Returns
True since a file IO object provides a seek method.

Return type
bool

tell()

Retrieves the current offset into the file input/output (IO) object.

dfvfs.vfs.apfs_container_directory module

The APFS container directory implementation.

class dfvfs.vfs.apfs_container_directory.**APFSContainerDirectory**(*file_system*, *path_spec*)

Bases: *Directory*

File system directory that uses pyfsapfs.

dfvfs.vfs.apfs_container_file_entry module

The APFS container file entry implementation.

class dfvfs.vfs.apfs_container_file_entry.**APFSContainerFileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*)

Bases: *FileEntry*

File system file entry that uses pyfsapfs.

GetAPFSVolume()

Retrieves an APFS volume.

Returns
an APFS volume or None if not available.

Return type
pyfsapfs.volume

GetParentFileEntry()

Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
APFSContainerFileEntry

IsLocked()

Determines if the file entry is locked.

Returns
True if the file entry is locked.

Return type
bool

TYPE_INDICATOR = 'APFS_CONTAINER'

Unlock()

Unlocks the file entry.

Returns
True if the file entry was unlocked.

Return type
bool

property name

name of the file entry, which does not include the full path.

Type
str

property size

size of the file entry in bytes or None if not available.

Type
int

property sub_file_entries

sub file entries.

Type
generator[*APFSContainerFileEntry*]

dfvfs.vfs.apfs_container_file_system module

The APFS container file system implementation.

class dfvfs.vfs.apfs_container_file_system.**APFSContainerFileSystem**(*resolver_context*, *path_spec*)

Bases: *FileSystem*

APFS container file system using pyfsapfs.

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – a path specification.

Returns
True if the file entry exists.

Return type
bool

GetAPFSContainer()

Retrieves the APFS container.

Returns
the APFS container.

Return type
pyfsapfs.container

GetAPFSVolumeByPathSpec(*path_spec*)

Retrieves an APFS volume for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
an APFS volume or None if not available.

Return type
pyfsapfs.volume

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
a file entry or None if not exists.

Return type
APFSContainerFileEntry

GetRootFileEntry()

Retrieves the root file entry.

Returns
a file entry.

Return type
APFSContainerFileEntry

TYPE_INDICATOR = 'APFS_CONTAINER'

dfvfs.vfs.apfs_directory module

The APFS directory implementation.

class dfvfs.vfs.apfs_directory.**APFSDirectory**(*file_system*, *path_spec*, *fsapfs_file_entry*)

Bases: *Directory*

File system directory that uses pyfsapfs.

dfvfs.vfs.apfs_file_entry module

The APFS file entry implementation.

class dfvfs.vfs.apfs_file_entry.**APFSFileEntry**(*resolver_context*, *file_system*, *path_spec*, *fsapfs_file_entry=None*, *is_root=False*, *is_virtual=False*)

Bases: *FileEntry*

File system file entry that uses pyfsapfs.

GetAPFSFileEntry()

Retrieves the APFS file entry.

Returns
APFS file entry.

Return type
pyfsapfs.file_entry

`GetExtents()`

Retrieves the extents.

Returns
the extents.

Return type
list[*Extent*]

`GetLinkedFileEntry()`

Retrieves the linked file entry, e.g. for a symbolic link.

Returns
linked file entry or None if not available.

Return type
APFSFileEntry

`GetParentFileEntry()`

Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
APFSFileEntry

`TYPE_INDICATOR = 'APFS'`

`property access_time`

access time or None if not available.

Type
dfdatetime.DateTimeValues

`property added_time`

added time or None if not available.

Type
dfdatetime.DateTimeValues

`property change_time`

change time or None if not available.

Type
dfdatetime.DateTimeValues

`property creation_time`

creation time or None if not available.

Type
dfdatetime.DateTimeValues

`property modification_time`

modification time or None if not available.

Type
dfdatetime.DateTimeValues

property name

name of the file entry, which does not include the full path.

Type
str

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.apfs_file_system module

The APFS file system implementation.

class dfvfs.vfs.apfs_file_system.**APFSFileSystem**(*resolver_context*, *path_spec*)

Bases: *FileSystem*

File system that uses pyfsapfs.

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the file entry exists.

Return type
bool

Raises
BackEndError – if the file entry cannot be opened.

GetAPFSFileEntryByPathSpec(*path_spec*)

Retrieves the APFS file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
file entry.

Return type
pyfsapfs.file_entry

Raises
PathSpecError – if the path specification is missing location and identifier.

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
file entry or None if not available.

Return type
APFSFileEntry

Raises
BackEndError – if the file entry cannot be opened.

GetRootFileEntry()

Retrieves the root file entry.

Returns
file entry or None if not available.

Return type
APFSFileEntry

ROOT_DIRECTORY_IDENTIFIER = 2

TYPE_INDICATOR = 'APFS'

dfvfs.vfs.attribute module

The Virtual File System (VFS) attribute interface.

class dfvfs.vfs.attribute.**Attribute**

Bases: object

Attribute interface.

GetExtents()

Retrieves the extents.

Returns
the extents of the attribute data.

Return type
list[*Extent*]

property type_indicator

type indicator or None if not known.

Type
str

class dfvfs.vfs.attribute.**StatAttribute**

Bases: object

Attribute that represents a POSIX stat.

device_number

major and minor device number (if block or character device file), derived from st_rdev.

Type
Tuple[int, int]

group_identifier

group identifier (GID), equivalent to st_gid.

Type
int

inode_number

number of the corresponding inode, equivalent to st_ino.

Type
int

mode

access mode, equivalent to st_mode.

Type
int

number_of_links

number of hard links, equivalent to st_nlink.

Type
int

owner_identifier

user identifier (UID) of the owner, equivalent to st_uid.

Type
int

size

size, in number of bytes, equivalent to st_size.

Type
int

type

file type, value derived from st_mode >> 12.

Type
str

TYPE_BLOCK_DEVICE = 'block_device'

TYPE_CHARACTER_DEVICE = 'character_device'

TYPE_DEVICE = 'device'

TYPE_DIRECTORY = 'directory'

TYPE_FILE = 'file'

TYPE_LINK = 'link'

TYPE_PIPE = 'pipe'

TYPE_SOCKET = 'socket'

TYPE_WHITEOUT = 'whiteout'

dfvfs.vfs.bde_file_entry module

The BDE file entry implementation.

class dfvfs.vfs.bde_file_entry.**BDEFileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*)

Bases: *RootOnlyFileEntry*

File system file entry that uses pybde.

IsLocked()

Determines if the file entry is locked.

Returns
True if the file entry is locked.

Return type
bool

TYPE_INDICATOR = 'BDE'

Unlock()

Unlocks the file entry.

Returns
True if the file entry was unlocked.

Return type
bool

property creation_time

creation time or None if not available.

Type
dfdatetime.DateTimeValues

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.bde_file_system module

The BDE file system implementation.

class dfvfs.vfs.bde_file_system.**BDEFileSystem**(*resolver_context*, *path_spec*)

Bases: *RootOnlyFileSystem*

File system that uses pybde.

GetBDEVolume()

Retrieves the BDE volume.

Returns
BDE volume.

Return type
pybde.volume

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
file entry or None.

Return type
BDEFileEntry

GetRootFileEntry()

Retrieves the root file entry.

Returns
file entry or None.

Return type
BDEFileEntry

TYPE_INDICATOR = 'BDE'

dfvfs.vfs.compressed_stream_file_entry module

The compressed stream file entry implementation.

class dfvfs.vfs.compressed_stream_file_entry.**CompressedStreamFileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*)

Bases: *RootOnlyFileEntry*

Compressed stream file entry.

TYPE_INDICATOR = 'COMPRESSED_STREAM'

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.compressed_stream_file_system module

The compressed stream file system implementation.

class dfvfs.vfs.compressed_stream_file_system.**CompressedStreamFileSystem**(*resolver_context*, *path_spec*)

Bases: *RootOnlyFileSystem*

Compressed stream file system.

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
a file entry or None if not available.

Return type
CompressedStreamFileEntry

`GetRootFileEntry()`
Retrieves the root file entry.

Returns
a file entry or None if not available.

Return type
CompressedStreamFileEntry

`TYPE_INDICATOR = 'COMPRESSED_STREAM'`

dfvfs.vfs.cpio_directory module

The CPIO directory implementation.

`class dfvfs.vfs.cpio_directory.CPIODirectory(file_system, path_spec)`
Bases: *Directory*

File system directory that uses CPIOArchiveFile.

dfvfs.vfs.cpio_file_entry module

The CPIO file entry implementation.

`class dfvfs.vfs.cpio_file_entry.CPIOFileEntry(resolver_context, file_system, path_spec, cpio_archive_file_entry=None, is_root=False, is_virtual=False)`
Bases: *FileEntry*

File system file entry that uses CPIOArchiveFile.

`GetCPIOArchiveFileEntry()`
Retrieves the CPIO archive file entry object.

Returns
CPIO archive file entry.

Return type
CPIOArchiveFileEntry

Raises
PathSpecError – if the path specification is incorrect.

`GetParentFileEntry()`
Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
CPIOFileEntry

`TYPE_INDICATOR = 'CPIO'`

property modification_time

modification time or None if not available.

Type
dfdatetime.DateTimeValues

property name

name of the file entry, which does not include the full path.

Type
str

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.cpio_file_system module

The CPIO archive file system implementation.

class dfvfs.vfs.cpio_file_system.**CPIOFileSystem**(*resolver_context*, *path_spec*, *encoding='utf-8'*)

Bases: *FileSystem*

CPIO archive file system.

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – a path specification.

Returns
True if the file entry exists.

Return type
bool

GetCPIOArchiveFile()

Retrieves the CPIO archive file.

Returns
a CPIO archive file.

Return type
CPIOArchiveFile

GetCPIOArchiveFileEntryByPathSpec(*path_spec*)

Retrieves the CPIO archive file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
CPIO archive file entry or None if not available.

Return type
CPIOArchiveFileEntry

Raises
PathSpecError – if the path specification is incorrect.

`GetFileEntryByPathSpec`(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
a file entry or None if not available.

Return type
CPIOFileEntry

`GetRootFileEntry`()

Retrieves the root file entry.

Returns
a file entry or None if not available.

Return type
CPIOFileEntry

`TYPE_INDICATOR = 'CPIO'`

dfvfs.vfs.cs_directory module

The Core Storage (CS) directory implementation.

`class dfvfs.vfs.cs_directory.CSDirectory`(*file_system*, *path_spec*)

Bases: *Directory*

File system directory that uses pyfvde.

dfvfs.vfs.cs_file_entry module

The Core Storage (CS) file entry implementation.

`class dfvfs.vfs.cs_file_entry.CSFileEntry`(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*, *fvde_logical_volume=None*)

Bases: *FileEntry*

File system file entry that uses pyfvde.

`GetFVDELogicalVolume`()

Retrieves the Core Storage logical volume.

Returns
a Core Storage logical volume.

Return type
pyfvde.logical_volume

`GetParentFileEntry`()

Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
CSFileEntry

`IsLocked()`

Determines if the file entry is locked.

Returns
True if the file entry is locked.

Return type
bool

`TYPE_INDICATOR = 'CS'`

`Unlock()`

Unlocks the file entry.

Returns
True if the file entry was unlocked.

Return type
bool

`property name`

name of the file entry, without the full path.

Type
str

`property size`

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.cs_file_system module

The Core Storage (CS) file system implementation.

`class dfvfs.vfs.cs_file_system.CSFileSystem(resolver_context, path_spec)`

Bases: *FileSystem*

File system that uses pyfvde.

`FileEntryExistsByPathSpec(path_spec)`

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the file entry exists.

Return type
bool

`GetFVDELogicalVolumeByPathSpec(path_spec)`

Retrieves a Core Storage logical volume for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns

a Core Storage logical volume or None if not
available.

Return type
pyfvde.logical_volume

`GetFVDEVolumeGroup()`

Retrieves the Core Storage volume group.

Returns
a Core Storage volume group.

Return type
pyfvde.volume_group

`GetFileEntryByPathSpec(path_spec)`

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
a file entry or None if not available.

Return type
CSFileEntry

`GetRootFileEntry()`

Retrieves the root file entry.

Returns
root file entry or None if not available.

Return type
CSFileEntry

`TYPE_INDICATOR = 'CS'`

dfvfs.vfs.data_range_file_entry module

The data range file entry implementation.

`class dfvfs.vfs.data_range_file_entry.DataRangeFileEntry(resolver_context, file_system, path_spec, is_root=False, is_virtual=False)`

Bases: *RootOnlyFileEntry*

File entry that represents a data range.

`TYPE_INDICATOR = 'DATA_RANGE'`

`property size`

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.data_range_file_system module

The data range file system implementation.

class dfvfs.vfs.data_range_file_system.**DataRangeFileSystem**(*resolver_context*, *path_spec*)

Bases: *RootOnlyFileSystem*

Data range file system.

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
a file entry or None if not available.

Return type
DataRangeFileEntry

GetRootFileEntry()

Retrieves the root file entry.

Returns
a file entry or None if not available.

Return type
DataRangeFileEntry

TYPE_INDICATOR = 'DATA_RANGE'

dfvfs.vfs.data_stream module

The Virtual File System (VFS) data stream interface.

class dfvfs.vfs.data_stream.**DataStream**(*file_entry*)

Bases: object

Data stream interface.

GetExtents()

Retrieves the extents.

Returns
the extents of the data stream.

Return type
list[*Extent*]

IsDefault()

Determines if the data stream is the default data stream.

Returns
True if the data stream is the default data stream.

Return type
bool

property name

name.

Type
str

dfvfs.vfs.directory module

The Virtual File System (VFS) directory interface.

class dfvfs.vfs.directory.**Directory**(*file_system*, *path_spec*)

Bases: object

Directory interface.

path_spec

path specification of the directory.

Type
PathSpec

property entries

path specifications of the directory entries.

Type
generator[*PathSpec*]

dfvfs.vfs.encoded_stream_file_entry module

The encoded stream file entry implementation.

class dfvfs.vfs.encoded_stream_file_entry.**EncodedStreamFileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*)

Bases: *RootOnlyFileEntry*

Class that implements an encoded stream file entry.

TYPE_INDICATOR = 'ENCODED_STREAM'

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.encoded_stream_file_system module

The encoded stream file system implementation.

class dfvfs.vfs.encoded_stream_file_system.**EncodedStreamFileSystem**(*resolver_context*, *path_spec*)

Bases: *RootOnlyFileSystem*

Encoded stream file system.

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
a file entry or None if not available.

Return type
EncodedStreamFileEntry

GetRootFileEntry()

Retrieves the root file entry.

Returns
a file entry or None if not available.

Return type
EncodedStreamFileEntry

TYPE_INDICATOR = 'ENCODED_STREAM'

dfvfs.vfs.encrypted_stream_file_entry module

The encrypted stream file entry implementation.

class dfvfs.vfs.encrypted_stream_file_entry.**EncryptedStreamFileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*)

Bases: *RootOnlyFileEntry*

Encrypted stream file entry.

TYPE_INDICATOR = 'ENCRYPTED_STREAM'

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.encrypted_stream_file_system module

The encrypted stream file system implementation.

class dfvfs.vfs.encrypted_stream_file_system.**EncryptedStreamFileSystem**(*resolver_context*, *path_spec*)

Bases: *RootOnlyFileSystem*

Encrypted stream file system.

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
a file entry or None if not available.

Return type
EncryptedStreamFileEntry

`GetRootFileEntry()`

Retrieves the root file entry.

Returns
a file entry or None if not available.

Return type
EncryptedStreamFileEntry

`TYPE_INDICATOR = 'ENCRYPTED_STREAM'`

dfvfs.vfs.ext_attribute module

The EXT attribute implementation.

`class dfvfs.vfs.ext_attribute.EXTExtendedAttribute(fsext_extended_attribute)`

Bases: *Attribute*

EXT extended attribute that uses pyfsext.

`GetExtents()`

Retrieves the extents.

Returns
the extents of the attribute data.

Return type
list[*Extent*]

`get_offset()`

Retrieves the current offset into the file input/output (IO) object.

Returns
current offset into the file input/output (IO) object.

Return type
int

`get_size()`

Retrieves the size of the file input/output (IO) object.

Returns
size of the file input/output (IO) object.

Return type
int

`property name`

name.

Type
str

read(*size=None*)

Reads a byte string from the file input/output (IO) object.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file input/output (IO) object.

Parameters

- **offset** (*int*) – offset to seek.
- **whence** (*Optional[int]*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

seekable()

Determines if a file input/output (IO) object is seekable.

Returns
True since a file IO object provides a seek method.

Return type
bool

tell()

Retrieves the current offset into the file input/output (IO) object.

dfvfs.vfs.ext_directory module

The EXT directory implementation.

class dfvfs.vfs.ext_directory.**EXTDirectory**(*file_system*, *path_spec*, *fsext_file_entry*)

Bases: *Directory*

File system directory that uses pyfsext.

dfvfs.vfs.ext_file_entry module

The EXT file entry implementation.

class dfvfs.vfs.ext_file_entry.**EXTFileEntry**(*resolver_context*, *file_system*, *path_spec*, *fsext_file_entry=None*, *is_root=False*, *is_virtual=False*)

Bases: *FileEntry*

File system file entry that uses pyfsext.

GetEXTFileEntry()

Retrieves the EXT file entry.

Returns
EXT file entry.

Return type
pyfsext.file_entry

GetExtents()

Retrieves the extents.

Returns
the extents.

Return type
list[*Extent*]

GetLinkedFileEntry()

Retrieves the linked file entry, e.g. for a symbolic link.

Returns
linked file entry or None if not available.

Return type
EXTFileEntry

GetParentFileEntry()

Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
EXTFileEntry

TYPE_INDICATOR = 'EXT'

property access_time

access time or None if not available.

Type
dfdatetime.DateTimeValues

property change_time

change time or None if not available.

Type
dfdatetime.DateTimeValues

property creation_time

creation time or None if not available.

Type
dfdatetime.DateTimeValues

property deletion_time

deletion time or None if not available.

Type
dfdatetime.DateTimeValues

property modification_time

modification time or None if not available.

Type
dfdatetime.DateTimeValues

property name

name of the file entry, which does not include the full path.

Type
str

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.ext_file_system module

The EXT file system implementation.

class dfvfs.vfs.ext_file_system.**EXTFileSystem**(*resolver_context*, *path_spec*)

Bases: *FileSystem*

File system that uses pyfsext.

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the file entry exists.

Return type
bool

Raises
BackEndError – if the file entry cannot be opened.

GetEXTFileEntryByPathSpec(*path_spec*)

Retrieves the EXT file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
file entry.

Return type
pyfsext.file_entry

Raises
PathSpecError – if the path specification is missing location and inode.

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
file entry or None if not available.

Return type
EXTFileEntry

Raises
BackEndError – if the file entry cannot be opened.

GetRootFileEntry()

Retrieves the root file entry.

Returns
file entry.

Return type
EXTFileEntry

ROOT_DIRECTORY_INODE_NUMBER = 2

TYPE_INDICATOR = 'EXT'

dfvfs.vfs.extent module

The Virtual File System (VFS) extent.

class dfvfs.vfs.extent.**Extent**(*extent_type=None*, *offset=None*, *size=None*)

Bases: object

Extent.

extent_type

type of the extent, for example EXTENT_TYPE_SPARSE.

Type
str

offset

offset of the extent relative from the start of the file system in bytes.

Type
int

size

size of the extent in bytes.

Type
int

dfvfs.vfs.fake_directory module

The fake directory implementation.

class dfvfs.vfs.fake_directory.**FakeDirectory**(*file_system*, *path_spec*)

Bases: *Directory*

Fake file system directory.

dfvfs.vfs.fake_file_entry module

The fake file entry implementation.

class dfvfs.vfs.fake_file_entry.**FakeFileEntry**(*resolver_context*, *file_system*, *path_spec*, *file_entry_type=None*, *is_root=False*)

Bases: *FileEntry*

Fake file system file entry.

GetFileObject(*data_stream_name=''*)

Retrieves a file-like object of a specific data stream.

Parameters
data_stream_name (*Optional[str]*) – name of the data stream, where an empty string represents the default data stream.

Returns
a file-like object or None if not available.

Return type
FakeFileIO

Raises

- **IOError** – if the file entry is not a file.
- **OSError** – if the file entry is not a file.

GetParentFileEntry()

Retrieves the root file entry.

Returns
parent file entry or None if not available.

Return type
FakeFileEntry

TYPE_INDICATOR = 'FAKE'

property access_time

access time or None if not available.

Type
dfdatetime.DateTimeValues

property change_time

change time or None if not available.

Type
dfdatetime.DateTimeValues

property modification_time

modification time or None if not available.

Type
dfdatetime.DateTimeValues

property name

name of the file entry, without the full path.

Type
str

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.fake_file_system module

The fake file system implementation.

class dfvfs.vfs.fake_file_system.**FakeFileSystem**(*resolver_context*, *path_spec*)

Bases: *FileSystem*

Fake file system.

AddFileEntry(*path*, *file_entry_type='file'*, *file_data=None*, *link_data=None*)

Adds a fake file entry.

Parameters

- **path** (*str*) – path of the file entry.
- **file_entry_type** (*Optional[str]*) – type of the file entry object.
- **file_data** (*Optional[bytes]*) – data of the fake file-like object.
- **link_data** (*Optional[bytes]*) – link data of the fake file entry object.

Raises

- **KeyError** – if the path already exists.
- **ValueError** – if the file data is set but the file entry type is not a file or if the link data is set but the file entry type is not a link.

FileEntryExistsByPath(*path*)

Determines if a file entry for a path exists.

Parameters
path (*str*) – path of the file entry.

Returns
True if the file entry exists.

Return type
bool

`FileEntryExistsByPathSpec`(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the file entry exists.

Return type
bool

`GetDataByPath`(*path*)

Retrieves the data associated to a path.

Parameters
path (*str*) – path of the file entry.

Returns
data or None if not available.

Return type
bytes

`GetFileEntryByPath`(*path*)

Retrieves a file entry for a path.

Parameters
path (*str*) – path of the file entry.

Returns
a file entry or None if not available.

Return type
FakeFileEntry

`GetFileEntryByPathSpec`(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
a file entry or None if not available.

Return type
FakeFileEntry

`GetPaths`()

Retrieves the paths dictionary.

Returns
file-like object per path.

Return type
dict[str, *FileIO*]

`GetRootFileEntry`()

Retrieves the root file entry.

Returns
a file entry or None if not available.

Return type
FakeFileEntry

`TYPE_INDICATOR = 'FAKE'`

dfvfs.vfs.fat_directory module

The FAT directory implementation.

`class dfvfs.vfs.fat_directory.FATDirectory(file_system, path_spec, fsfat_file_entry)`

Bases: *Directory*

File system directory that uses pyfsfat.

dfvfs.vfs.fat_file_entry module

The FAT file entry implementation.

`class dfvfs.vfs.fat_file_entry.FATFileEntry(resolver_context, file_system, path_spec, fsfat_file_entry=None, is_root=False, is_virtual=False)`

Bases: *FileEntry*

File system file entry that uses pyfsfat.

`GetExtents()`

Retrieves the extents.

Returns
the extents.

Return type
list[*Extent*]

`GetFATFileEntry()`

Retrieves the FAT file entry.

Returns
FAT file entry.

Return type
pyfsfat.file_entry

`GetFileObject(data_stream_name='')`

Retrieves a file-like object of a specific data stream.

Parameters
data_stream_name (*Optional[str]*) – name of the data stream, where an empty string represents the default data stream.

Returns
a file-like object or None if not available.

Return type
FileIO

GetParentFileEntry()

Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
FATFileEntry

TYPE_INDICATOR = 'FAT'

property access_time

access time or None if not available.

Type
dfdatetime.DateTimeValues

property creation_time

creation time or None if not available.

Type
dfdatetime.DateTimeValues

property modification_time

modification time or None if not available.

Type
dfdatetime.DateTimeValues

property name

name of the file entry, which does not include the full path.

Type
str

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.fat_file_system module

The FAT file system implementation.

class dfvfs.vfs.fat_file_system.**FATFileSystem**(*resolver_context*, *path_spec*)

Bases: *FileSystem*

File system that uses pyfsfat.

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the file entry exists.

Return type
bool

Raises
BackEndError – if the file entry cannot be opened.

GetFATFileEntryByPathSpec(*path_spec*)

Retrieves the FAT file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
file entry.

Return type
pyfsfat.file_entry

Raises
PathSpecError – if the path specification is missing location and identifier.

GetFATVolume()

Retrieves the FAT volume.

Returns
a FAT volume.

Return type
pyfsfat.volume

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
file entry or None if not available.

Return type
FATFileEntry

Raises
BackEndError – if the file entry cannot be opened.

GetRootFileEntry()

Retrieves the root file entry.

Returns
file entry.

Return type
FATFileEntry

LOCATION_ROOT = '\\'

PATH_SEPARATOR = '\\'

TYPE_INDICATOR = 'FAT'

dfvfs.vfs.file_entry module

The Virtual File System (VFS) file entry interface.

The file entry can be various file system elements like a regular file, a directory or file system metadata.

class dfvfs.vfs.file_entry.**FileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*)

Bases: `object`

File entry interface.

entry_type

file entry type, such as device, directory, file, link, socket and pipe or None if not available. The available file entry types are defined in dfvfs.lib.definitions for example FILE_ENTRY_TYPE_FILE.

Type
str

path_spec

path specification.

Type
PathSpec

GetDataStream(*name*, *case_sensitive=True*)

Retrieves a data stream by name.

Parameters

- **name** (*str*) – name of the data stream.
- **case_sensitive** (*Optional[bool]*) – True if the name is case sensitive.

Returns
a data stream or None if not available.

Return type
DataStream

Raises
ValueError – if the name is not string.

GetExtents()

Retrieves the extents.

Returns
the extents.

Return type
list[*Extent*]

GetFileObject(*data_stream_name=''*)

Retrieves a file-like object of a specific data stream.

Parameters
data_stream_name (*Optional[str]*) – name of the data stream, where an empty string represents the default data stream.

Returns
a file-like object or None if not available.

Return type
FileIO

`GetFileSystem`()

Retrieves the file system which contains the file entry.

Returns
a file system.

Return type
FileSystem

`GetLinkedFileEntry`()

Retrieves the linked file entry, for example for a symbolic link.

Returns
linked file entry or None if not available.

Return type
FileEntry

`GetParentFileEntry`()

Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
FileEntry

`GetStatAttribute`()

Retrieves a stat attribute.

Returns
a stat attribute or None if not available.

Return type
StatAttribute

`GetSubFileEntryByName`(*name*, *case_sensitive=True*)

Retrieves a sub file entry by name.

Parameters

- **name** (*str*) – name of the file entry.
- **case_sensitive** (*Optional[bool]*) – True if the name is case sensitive.

Returns
a file entry or None if not available.

Return type
FileEntry

`HasDataStream`(*name*, *case_sensitive=True*)

Determines if the file entry has specific data stream.

Parameters

- **name** (*str*) – name of the data stream.
- **case_sensitive** (*Optional[bool]*) – True if the name is case sensitive.

Returns
True if the file entry has the data stream.

Return type
bool

Raises
ValueError – if the name is not string.

`HasExternalData()`

Determines if the file entry has external stored data.

Returns
True if the file entry has external stored data.

Return type
bool

`IsAllocated()`

Determines if the file entry is allocated.

Returns
True if the file entry is allocated.

Return type
bool

`IsDevice()`

Determines if the file entry is a device.

Returns
True if the file entry is a device.

Return type
bool

`IsDirectory()`

Determines if the file entry is a directory.

Returns
True if the file entry is a directory.

Return type
bool

`IsFile()`

Determines if the file entry is a file.

Returns
True if the file entry is a file.

Return type
bool

`IsLink()`

Determines if the file entry is a link.

Returns
True if the file entry is a link.

Return type
bool

`IsLocked()`

Determines if the file entry is locked.

Returns
True if the file entry is locked.

Return type
bool

`IsPipe()`

Determines if the file entry is a pipe.

Returns
True if the file entry is a pipe.

Return type
bool

`IsRoot()`

Determines if the file entry is the root file entry.

Returns
True if the file entry is the root file entry.

Return type
bool

`IsSocket()`

Determines if the file entry is a socket.

Returns
True if the file entry is a socket.

Return type
bool

`IsVirtual()`

Determines if the file entry is virtual (emulated by dfVFS).

Returns
True if the file entry is virtual.

Return type
bool

`Unlock()`

Unlocks the file entry.

Returns
True if the file entry was unlocked.

Return type
bool

`property access_time`

access time or None if not available.

Type
dfdatetime.DateTimeValues

property added_time
added time or None if not available.

Type
dfdatetime.DateTimeValues

property attributes
attributes.

Type
generator[*Attribute*]

property backup_time
backup time or None if not available.

Type
dfdatetime.DateTimeValues

property change_time
change time or None if not available.

Type
dfdatetime.DateTimeValues

property creation_time
creation time or None if not available.

Type
dfdatetime.DateTimeValues

property data_streams
data streams.

Type
generator[*DataStream*]

property deletion_time
deletion time or None if not available.

Type
dfdatetime.DateTimeValues

property link
full path of the linked file entry or None if not available.

Type
str

property modification_time
modification time or None if not available.

Type
dfdatetime.DateTimeValues

abstract property name
name of the file entry, without the full path.

Type
str

property number_of_attributes

number of attributes.

Type
int

property number_of_data_streams

number of data streams.

Type
int

property number_of_sub_file_entries

number of sub file entries.

Type
int

property size

size of the file entry in bytes or None if not available.

Type
int

property sub_file_entries

sub file entries.

Type
generator[*FileEntry*]

property type_indicator

type indicator.

Type
str

dfvfs.vfs.file_system module

The Virtual File System (VFS) file system interface.

class dfvfs.vfs.file_system.**FileSystem**(*resolver_context*, *path_spec*)

Bases: object

File system interface.

BasenamePath(*path*)

Determines the basename of the path.

Parameters
path (*str*) – path.

Returns
basename of the path.

Return type
str

DirnamePath(*path*)

Determines the directory name of the path.

The file system root is represented by an empty string.

Parameters
path (*str*) – path.

Returns
directory name of the path or None.

Return type
str

abstract FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – a path specification.

Returns
True if the file entry exists.

Return type
bool

GetDataStreamByPathSpec(*path_spec*)

Retrieves a data stream for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
a data stream or None if not available.

Return type
DataStream

abstract GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
a file entry or None if not available.

Return type
FileEntry

GetFileObjectByPathSpec(*path_spec*)

Retrieves a file-like object for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
a file-like object or None if not available.

Return type
FileIO

GetPathSegmentAndSuffix(*base_path*, *path*)

Determines the path segment and suffix of the path.

None is returned if the path does not start with the base path and an empty string if the path exactly matches the base path.

Parameters

- **base_path** (*str*) – base path.
- **path** (*str*) – path.

Returns
path segment and suffix string.

Return type
tuple[str, str]

abstract GetRootFileEntry()

Retrieves the root file entry.

Returns
a file entry or None if not available.

Return type
FileEntry

JoinPath(*path_segments*)

Joins the path segments into a path.

Parameters
path_segments (*list[str]*) – path segments.

Returns
joined path segments prefixed with the path separator.

Return type
str

LOCATION_ROOT = '/'

Open(*path_spec=None*, *mode='rb'*)

Opens the file system object defined by path specification.

Parameters

- **path_spec** (*Optional[PathSpec]*) – a path specification.
- **mode** (*Optional[str]*) – file access mode. The default is 'rb' which represents read-only binary.

Raises

- ***AccessError*** – if the access to open the file was denied.
- **IOError** – if the file system object was already opened or the open failed.
- **OSError** – if the file system object was already opened or the open failed.
- ***PathSpecError*** – if the path specification is incorrect.
- **ValueError** – if the path specification or mode is invalid.

PATH_SEPARATOR = '/'

SplitPath(*path*)

Splits the path into path segments.

Parameters
path (*str*) – path.

Returns

path segments without the root path segment, which is
an empty string.

Return type
list[str]

`__del__()`
Cleans up the file system.

`property type_indicator`
type indicator.

Type
str

dfvfs.vfs.gpt_directory module

The GUID Partition Table (GPT) directory implementation.

`class dfvfs.vfs.gpt_directory.GPTDirectory(file_system, path_spec)`
Bases: *Directory*

File system directory that uses pyvsgpt.

dfvfs.vfs.gpt_file_entry module

The GUID Partition Table (GPT) file entry implementation.

`class dfvfs.vfs.gpt_file_entry.GPTFileEntry(resolver_context, file_system, path_spec, is_root=False, is_virtual=False, vsgpt_partition=None)`
Bases: *FileEntry*

File system file entry that uses pyvsgpt.

`GetGPTPartition()`
Retrieves the GPT partition.

Returns
a GPT partition.

Return type
pyvsgpt.partition

`GetParentFileEntry()`
Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
GPTFileEntry

`TYPE_INDICATOR = 'GPT'`

`property name`
name of the file entry, without the full path.

Type
str

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.gpt_file_system module

The GUID Partition Table (GPT) file system implementation.

class dfvfs.vfs.gpt_file_system.**GPTFileSystem**(*resolver_context*, *path_spec*)

Bases: *FileSystem*

File system that uses pyvsgpt.

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the file entry exists.

Return type
bool

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
a file entry or None if not available.

Return type
GPTFileEntry

GetGPTPartitionByPathSpec(*path_spec*)

Retrieves a GPT partition for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
a GPT partition or None if not available.

Return type
pyvsgpt.partition

GetGPTVolume()

Retrieves the GPT volume.

Returns
a GPT volume.

Return type
pyvsgpt.volume

GetRootFileEntry()

Retrieves the root file entry.

Returns
root file entry or None if not available.

Return type
GPTFileEntry

TYPE_INDICATOR = 'GPT'

dfvfs.vfs.gzip_file_entry module

The gzip file entry implementation.

class dfvfs.vfs.gzip_file_entry.**GzipFileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*)

Bases: *RootOnlyFileEntry*

File system file entry that uses gzip.

TYPE_INDICATOR = 'GZIP'

property modification_time

modification time or None if not available.

Type
dfdatetime.DateTimeValues

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.gzip_file_system module

The gzip file system implementation.

class dfvfs.vfs.gzip_file_system.**GzipFileSystem**(*resolver_context*, *path_spec*)

Bases: *RootOnlyFileSystem*

File system that uses gzip.

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
a file entry or None if not available.

Return type
GzipFileEntry

GetRootFileEntry()

Retrieves the root file entry.

Returns
a file entry or None if not available.

Return type
GzipFileEntry

TYPE_INDICATOR = 'GZIP'

dfvfs.vfs.hfs_attribute module

The HFS attribute implementation.

class dfvfs.vfs.hfs_attribute.**HFSExtendedAttribute**(*fshfs_extended_attribute*)

Bases: *Attribute*

HFS extended attribute that uses pyfshfs.

GetExtents()

Retrieves the extents.

Returns
the extents of the attribute data.

Return type
list[*Extent*]

get_offset()

Retrieves the current offset into the file input/output (IO) object.

Returns
current offset into the file input/output (IO) object.

Return type
int

get_size()

Retrieves the size of the file input/output (IO) object.

Returns
size of the file input/output (IO) object.

Return type
int

property name

name.

Type
str

read(*size=None*)

Reads a byte string from the file input/output (IO) object.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file input/output (IO) object.

Parameters

- **offset** (*int*) – offset to seek.
- **whence** (*Optional[int]*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

seekable()

Determines if a file input/output (IO) object is seekable.

Returns
True since a file IO object provides a seek method.

Return type
bool

tell()

Retrieves the current offset into the file input/output (IO) object.

dfvfs.vfs.hfs_data_stream module

The HFS data stream implementation.

class dfvfs.vfs.hfs_data_stream.**HFSDataStream**(*file_entry*, *fshfs_data_stream*)

Bases: *DataStream*

File system data stream that uses pyfshfs.

GetExtents()

Retrieves the extents.

Returns
the extents of the data stream.

Return type
list[*Extent*]

dfvfs.vfs.hfs_directory module

The HFS directory implementation.

class dfvfs.vfs.hfs_directory.**HFSDirectory**(*file_system*, *path_spec*, *fshfs_file_entry*)

Bases: *Directory*

File system directory that uses pyfshfs.

dfvfs.vfs.hfs_file_entry module

The HFS file entry implementation.

class dfvfs.vfs.hfs_file_entry.**HFSFileEntry**(*resolver_context*, *file_system*, *path_spec*, *fshfs_file_entry=None*, *is_root=False*, *is_virtual=False*)

Bases: *FileEntry*

File system file entry that uses pyfshfs.

GetExtents()

Retrieves the extents.

Returns
the extents.

Return type
list[*Extent*]

GetFileObject(*data_stream_name=''*)

Retrieves a file-like object of a specific data stream.

Parameters
data_stream_name (*Optional[str]*) – name of the data stream, where an empty string represents the default data stream.

Returns
a file-like object or None if not available.

Return type
FileIO

GetHFSFileEntry()

Retrieves the HFS file entry.

Returns
HFS file entry.

Return type
pyfshfs.file_entry

GetLinkedFileEntry()

Retrieves the linked file entry, e.g. for a symbolic link.

Returns
linked file entry or None if not available.

Return type
HFSFileEntry

GetParentFileEntry()

Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
HFSFileEntry

TYPE_INDICATOR = 'HFS'

property access_time

access time or None if not available.

Type
dfdatetime.DateTimeValues

property added_time

added time or None if not available.

Type
dfdatetime.DateTimeValues

property backup_time

backup time or None if not available.

Type
dfdatetime.DateTimeValues

property change_time

change time or None if not available.

Type
dfdatetime.DateTimeValues

property creation_time

creation time or None if not available.

Type
dfdatetime.DateTimeValues

property modification_time

modification time or None if not available.

Type
dfdatetime.DateTimeValues

property name

name of the file entry, which does not include the full path.

Type
str

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.hfs_file_system module

The HFS file system implementation.

class dfvfs.vfs.hfs_file_system.**HFSFileSystem**(*resolver_context*, *path_spec*)

Bases: *FileSystem*

File system that uses pyfshfs.

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the file entry exists.

Return type
bool

Raises
BackEndError – if the file entry cannot be opened.

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
file entry or None if not available.

Return type
HFSFileEntry

Raises
BackEndError – if the file entry cannot be opened.

GetHFSFileEntryByPathSpec(*path_spec*)

Retrieves the HFS file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
file entry.

Return type
pyfshfs.file_entry

Raises
PathSpecError – if the path specification is missing location and identifier.

GetRootFileEntry()

Retrieves the root file entry.

Returns
file entry.

Return type
HFSFileEntry

ROOT_DIRECTORY_IDENTIFIER_NUMBER = 2

TYPE_INDICATOR = 'HFS'

dfvfs.vfs.luksde_file_entry module

The LUKSDE file entry implementation.

class dfvfs.vfs.luksde_file_entry.**LUKSDEFileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*)

Bases: *RootOnlyFileEntry*

File system file entry that uses pyluksde.

IsLocked()

Determines if the file entry is locked.

Returns
True if the file entry is locked.

Return type
bool

TYPE_INDICATOR = 'LUKSDE'

Unlock()

Unlocks the file entry.

Returns
True if the file entry was unlocked.

Return type
bool

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.luksde_file_system module

The LUKSDE file system implementation.

class dfvfs.vfs.luksde_file_system.**LUKSDEFileSystem**(*resolver_context*, *path_spec*)

Bases: *RootOnlyFileSystem*

File system that uses pyluksde.

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
file entry or None.

Return type
LUKSDEFileEntry

GetLUKSDEVolume()

Retrieves the LUKSDE volume.

Returns
LUKSDE volume.

Return type
pyluksde.volume

GetRootFileEntry()

Retrieves the root file entry.

Returns
file entry or None.

Return type
LUKSDEFileEntry

TYPE_INDICATOR = 'LUKSDE'

dfvfs.vfs.lvm_directory module

The Logical Volume Manager (LVM) directory implementation.

class dfvfs.vfs.lvm_directory.**LVMDirectory**(*file_system*, *path_spec*)

Bases: *Directory*

File system directory that uses pyvslvm.

dfvfs.vfs.lvm_file_entry module

The Logical Volume Manager (LVM) file entry implementation.

class dfvfs.vfs.lvm_file_entry.**LVMFileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*, *vslvm_logical_volume=None*)

Bases: *FileEntry*

File system file entry that uses pyvslvm.

GetLVMLogicalVolume()

Retrieves the LVM logical volume.

Returns
a LVM logical volume.

Return type
pyvslvm.logical_volume

GetParentFileEntry()

Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
LVMFileEntry

TYPE_INDICATOR = 'LVM'

property name

name of the file entry, without the full path.

Type
str

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.lvm_file_system module

The Logical Volume Manager (LVM) file system implementation.

class dfvfs.vfs.lvm_file_system.**LVMFileSystem**(*resolver_context*, *path_spec*)

Bases: *FileSystem*

File system that uses pyvslvm.

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the file entry exists.

Return type
bool

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
a file entry or None if not available.

Return type
LVMFileEntry

GetLVMLogicalVolumeByPathSpec(*path_spec*)

Retrieves a LVM logical volume for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
a LVM logical volume or None if not available.

Return type
pyvslvm.logical_volume

GetLVMVolumeGroup()

Retrieves the LVM volume group.

Returns
a LVM volume group.

Return type
pyvslvm.volume_group

`GetRootFileEntry()`

Retrieves the root file entry.

Returns
root file entry or None if not available.

Return type
LVMFileEntry

`TYPE_INDICATOR = 'LVM'`

dfvfs.vfs.ntfs_attribute module

The NTFS attribute implementations.

`class dfvfs.vfs.ntfs_attribute.FileNameNTFSAttribute(fsntfs_attribute)`

Bases: *NTFSAttribute*

NTFS $FILE_NAME file system attribute.

`TYPE_INDICATOR = 'NTFS:$FILE_NAME'`

`property access_time`

access time or None if not set.

Type
dfdatetime.Filetime

`property creation_time`

creation time or None if not set.

Type
dfdatetime.Filetime

`property entry_modification_time`

entry modification time or None if not set.

Type
dfdatetime.Filetime

`property file_attribute_flags`

file attribute flags or None if not available.

Type
int

`property modification_time`

modification time.

Type
dfdatetime.Filetime

property name

name.

Type
str

property name_space

name_space.

Type
int

property parent_file_reference

parent file reference.

Type
int

class dfvfs.vfs.ntfs_attribute.**NTFSAttribute**(*fsntfs_attribute*)

Bases: *Attribute*

File system attribute that uses pyfsntfs.

property attribute_type

The attribute type.

class dfvfs.vfs.ntfs_attribute.**ObjectIdentifierNTFSAttribute**(*fsntfs_attribute*)

Bases: *NTFSAttribute*

NTFS $OBJECT_ID file system attribute.

TYPE_INDICATOR = 'NTFS:$OBJECT_ID'

property droid_file_identifier

droid file identifier, formatted as an UUID.

Type
str

class dfvfs.vfs.ntfs_attribute.**SecurityDescriptorNTFSAttribute**(*fsntfs_attribute*)

Bases: *NTFSAttribute*

NTFS $SECURITY_DESCRIPTOR file system attribute.

TYPE_INDICATOR = 'NTFS:$SECURITY_DESCRIPTOR'

property security_descriptor

security descriptor.

Type
pyfwnt.security_descriptor

class dfvfs.vfs.ntfs_attribute.**StandardInformationNTFSAttribute**(*fsntfs_attribute*)

Bases: *NTFSAttribute*

NTFS $STANDARD_INFORMATION file system attribute.

TYPE_INDICATOR = 'NTFS:$STANDARD_INFORMATION'

property access_time

access time or None if not set.

Type
dfdatetime.Filetime

property creation_time

creation time or None if not set.

Type
dfdatetime.Filetime

property entry_modification_time

entry modification time or None if not set.

Type
dfdatetime.Filetime

property file_attribute_flags

file attribute flags or None if not available.

Type
int

property modification_time

modification time or None if not set.

Type
dfdatetime.Filetime

property owner_identifier

owner identifier.

Type
int

property security_descriptor_identifier

security descriptor identifier.

Type
int

property update_sequence_number

update sequence number.

Type
int

dfvfs.vfs.ntfs_data_stream module

The NTFS data stream implementation.

class dfvfs.vfs.ntfs_data_stream.**NTFSDataStream**(*file_entry*, *fsntfs_data_stream*)

Bases: *DataStream*

File system data stream that uses pyfsntfs.

GetExtents()

Retrieves the extents.

Returns
the extents of the data stream.

Return type
list[*Extent*]

dfvfs.vfs.ntfs_directory module

The NTFS directory implementation.

class dfvfs.vfs.ntfs_directory.**NTFSDirectory**(*file_system*, *path_spec*, *fsntfs_file_entry*)

Bases: *Directory*

File system directory that uses pyfsntfs.

dfvfs.vfs.ntfs_file_entry module

The NTFS file entry implementation.

class dfvfs.vfs.ntfs_file_entry.**NTFSFileEntry**(*resolver_context*, *file_system*, *path_spec*, *fsntfs_file_entry=None*, *is_root=False*, *is_virtual=False*)

Bases: *FileEntry*

File system file entry that uses pyfsntfs.

GetExtents()

Retrieves the extents.

Returns
the extents.

Return type
list[*Extent*]

GetFileObject(*data_stream_name=''*)

Retrieves a file-like object of a specific data stream.

Parameters
data_stream_name (*Optional[str]*) – data stream name, where an empty string represents the default data stream.

Returns
file-like object or None.

Return type
NTFSFileIO

GetLinkedFileEntry()

Retrieves the linked file entry, e.g. for a symbolic link.

Returns
linked file entry or None.

Return type
NTFSFileEntry

`GetNTFSFileEntry()`

Retrieves the NTFS file entry.

Returns
NTFS file entry.

Return type
pyfsntfs.file_entry

`GetParentFileEntry()`

Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
NTFSFileEntry

`GetSecurityDescriptor()`

Retrieves the security descriptor.

Returns
security descriptor.

Return type
pyfwnt.security_descriptor

`IsAllocated()`

Determines if the file entry is allocated.

Returns
True if the file entry is allocated.

Return type
bool

`TYPE_INDICATOR = 'NTFS'`

`property access_time`

access time or None if not available.

Type
dfdatetime.DateTimeValues

`property change_time`

change time or None if not available.

Type
dfdatetime.DateTimeValues

`property creation_time`

creation time or None if not available.

Type
dfdatetime.DateTimeValues

`property modification_time`

modification time or None if not available.

Type
dfdatetime.DateTimeValues

property name

name of the file entry, which does not include the full path.

Type
str

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.ntfs_file_system module

The NTFS file system implementation.

class dfvfs.vfs.ntfs_file_system.**NTFSFileSystem**(*resolver_context*, *path_spec*)

Bases: *FileSystem*

File system that uses pyfsntfs.

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the file entry exists.

Return type
bool

Raises
BackEndError – if the file entry cannot be opened.

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
file entry or None if not available.

Return type
NTFSFileEntry

Raises
BackEndError – if the file entry cannot be opened.

GetNTFSFileEntryByPathSpec(*path_spec*)

Retrieves the NTFS file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
NTFS file entry.

Return type
pyfsntfs.file_entry

Raises
PathSpecError – if the path specification is missing location and MFT entry.

`GetRootFileEntry()`

Retrieves the root file entry.

Returns
file entry.

Return type
NTFSFileEntry

`LOCATION_ROOT = '\\'`

`MFT_ENTRY_ROOT_DIRECTORY = 5`

`PATH_SEPARATOR = '\\'`

`TYPE_INDICATOR = 'NTFS'`

dfvfs.vfs.os_attribute module

The operating system attribute implementation.

`class dfvfs.vfs.os_attribute.OSExtendedAttribute(location, name)`

Bases: *Attribute*

Extended attribute that uses the operating system.

`get_offset()`

Retrieves the current offset into the file input/output (IO) object.

Returns
current offset into the file input/output (IO) object.

Return type
int

`get_size()`

Retrieves the size of the file input/output (IO) object.

Returns
size of the file input/output (IO) object.

Return type
int

`property name`

name.

Type
str

`read(size=None)`

Reads a byte string from the file input/output (IO) object.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file input/output (IO) object.

Parameters

- **offset** (*int*) – offset to seek.
- **whence** (*Optional[int]*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

seekable()

Determines if a file input/output (IO) object is seekable.

Returns
True since a file IO object provides a seek method.

Return type
bool

tell()

Retrieves the current offset into the file input/output (IO) object.

dfvfs.vfs.os_directory module

The operating system directory implementation.

class dfvfs.vfs.os_directory.**OSDirectory**(*file_system*, *path_spec*)

Bases: *Directory*

File system directory that uses the operating system.

dfvfs.vfs.os_file_entry module

The operating system file entry implementation.

class dfvfs.vfs.os_file_entry.**OSFileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*)

Bases: *FileEntry*

File system file entry that uses os.

GetLinkedFileEntry()

Retrieves the linked file entry, for example for a symbolic link.

Returns
linked file entry or None if not available.

Return type
OSFileEntry

GetParentFileEntry()

Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
OSFileEntry

TYPE_INDICATOR = 'OS'

property access_time

access time or None if not available.

Type
dfdatetime.DateTimeValues

property change_time

change time or None if not available.

Type
dfdatetime.DateTimeValues

property creation_time

creation time or None if not available.

Type
dfdatetime.DateTimeValues

property modification_time

modification time or None if not available.

Type
dfdatetime.DateTimeValues

property name

name of the file entry, without the full path.

Type
str

property size
size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.os_file_system module

The operating system file system implementation.

class dfvfs.vfs.os_file_system.**OSFileSystem**(*resolver_context*, *path_spec*)

Bases: *FileSystem*

File system that uses the operating system.

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – a path specification.

Returns
True if the file entry exists, false otherwise.

Return type
bool

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
a file entry or None if not available.

Return type
OSFileEntry

GetRootFileEntry()

Retrieves the root file entry.

Returns
a file entry or None if not available.

Return type
OSFileEntry

JoinPath(*path_segments*)

Joins the path segments into a path.

Parameters
path_segments (*list[str]*) – path segments.

Returns
joined path segments prefixed with the path separator.

Return type
str

PATH_SEPARATOR = '/'

TYPE_INDICATOR = 'OS'

dfvfs.vfs.root_only_file_entry module

The root only file system file entry implementation.

class dfvfs.vfs.root_only_file_entry.**RootOnlyFileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*)

Bases: *FileEntry*

Root only file system file entry.

property name

name of the file entry, without the full path.

Type
str

dfvfs.vfs.root_only_file_system module

The root only file system implementation.

class dfvfs.vfs.root_only_file_system.**RootOnlyFileSystem**(*resolver_context*, *path_spec*)

Bases: *FileSystem*

Root only file system.

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the file entry exists.

Return type
bool

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
a file entry or None if not available.

Return type
FileEntry

abstract GetRootFileEntry()

Retrieves the root file entry.

Returns
a file entry or None if not available.

Return type
FileEntry

dfvfs.vfs.sqlite_blob_directory module

The SQLite blob directory implementation.

class dfvfs.vfs.sqlite_blob_directory.**SQLiteBlobDirectory**(*file_system*, *path_spec*)

Bases: *Directory*

SQLite blob directory.

dfvfs.vfs.sqlite_blob_file_entry module

The SQLite blob file entry implementation.

class dfvfs.vfs.sqlite_blob_file_entry.**SQLiteBlobFileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*)

Bases: *FileEntry*

SQLite blob file entry.

GetNumberOfRows()

Retrieves the number of rows in the table.

Returns
number of rows.

Return type
int

Raises
BackEndError – when the SQLite blob file-like object is missing.

GetParentFileEntry()

Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
SQLiteBlobFileEntry

TYPE_INDICATOR = 'SQLITE_BLOB'

property name

name of the file entry, which does not include the full path.

Type
str

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.sqlite_blob_file_system module

The SQLite blob file system implementation.

class dfvfs.vfs.sqlite_blob_file_system.**SQLiteBlobFileSystem**(*resolver_context*, *path_spec*)

Bases: *FileSystem*

Class that implements a file system object using SQLite blob.

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the file entry exists.

Return type
bool

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
a file entry or None.

Return type
FileEntry

GetRootFileEntry()

Retrieves the root file entry.

Returns
a file entry or None.

Return type
FileEntry

TYPE_INDICATOR = 'SQLITE_BLOB'

dfvfs.vfs.tar_directory module

The TAR directory implementation.

class dfvfs.vfs.tar_directory.**TARDirectory**(*file_system*, *path_spec*)

Bases: *Directory*

File system directory that uses tarfile.

dfvfs.vfs.tar_file_entry module

The TAR file entry implementation.

class dfvfs.vfs.tar_file_entry.**TARFileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*, *tar_info=None*)

Bases: *FileEntry*

File system file entry that uses tarfile.

GetParentFileEntry()

Retrieves the parent file entry.

Returns
parent file entry or None.

Return type
TARFileEntry

GetTARInfo()

Retrieves the TAR info.

Returns
TAR info or None if it does not exist.

Return type
tarfile.TARInfo

Raises
PathSpecError – if the path specification is incorrect.

TYPE_INDICATOR = 'TAR'

property modification_time

modification time or None if not available.

Type
dfdatetime.DateTimeValues

property name

name of the file entry, which does not include the full path.

Type
str

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.tar_file_system module

The TAR file system implementation.

class dfvfs.vfs.tar_file_system.**TARFileSystem**(*resolver_context*, *path_spec*, *encoding='utf-8'*)

Bases: *FileSystem*

Class that implements a file system using tarfile.

encoding

file entry name encoding.

Type
str

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the file entry exists.

Return type
bool

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
file entry or None.

Return type
TARFileEntry

GetRootFileEntry()

Retrieves the root file entry.

Returns
file entry.

Return type
TARFileEntry

GetTARFile()

Retrieves the TAR file.

Returns
TAR file.

Return type
tarfile.TARFile

GetTARInfoByPathSpec(*path_spec*)

Retrieves the TAR info for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
TAR info or None if it does not exist.

Return type
tarfile.TARInfo

Raises
PathSpecError – if the path specification is incorrect.

`TYPE_INDICATOR = 'TAR'`

dfvfs.vfs.tsk_attribute module

The SleuthKit (TSK) attribute implementation.

`class dfvfs.vfs.tsk_attribute.TSKAttribute(tsk_file, tsk_attribute)`

Bases: *Attribute*

File system attribute that uses pytsk3.

`property attribute_type`

attribute type.

Type
object

`class dfvfs.vfs.tsk_attribute.TSKExtendedAttribute(tsk_file, tsk_attribute)`

Bases: *TSKAttribute*

File system extended attribute that uses pytsk3.

`get_offset()`

Retrieves the current offset into the file input/output (IO) object.

Returns
current offset into the file input/output (IO) object.

Return type
int

`get_size()`

Retrieves the size of the file input/output (IO) object.

Returns
size of the file input/output (IO) object.

Return type
int

`property name`

name.

Raises
BackEndError – if pytsk3 returns a non UTF-8 formatted name.

Type
str

read(*size=None*)

Reads a byte string from the file input/output (IO) object.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file input/output (IO) object.

Parameters

- **offset** (*int*) – offset to seek.
- **whence** (*Optional[int]*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

seekable()

Determines if a file input/output (IO) object is seekable.

Returns
True since a file IO object provides a seek method.

Return type
bool

tell()

Retrieves the current offset into the file input/output (IO) object.

dfvfs.vfs.tsk_data_stream module

The SleuthKit (TSK) data stream implementation.

class dfvfs.vfs.tsk_data_stream.**TSKDataStream**(*file_entry*, *pytsk_attribute*)

Bases: *DataStream*

File system data stream that uses pytsk3.

GetExtents()

Retrieves the extents.

Returns
the extents of the data stream.

Return type
list[*Extent*]

Raises
BackEndError – if pytsk3 returns no file system block size or data stream size.

dfvfs.vfs.tsk_directory module

The SleuthKit (TSK) directory implementation.

class dfvfs.vfs.tsk_directory.**TSKDirectory**(*file_system*, *path_spec*)

Bases: *Directory*

File system directory that uses pytsk3.

dfvfs.vfs.tsk_file_entry module

The SleuthKit (TSK) file entry implementation.

class dfvfs.vfs.tsk_file_entry.**TSKFileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*, *parent_inode=None*, *tsk_file=None*)

Bases: *FileEntry*

File system file entry that uses pytsk3.

GetExtents()

Retrieves the extents.

Returns
the extents.

Return type
list[*Extent*]

Raises
BackEndError – if pytsk3 returns no file system block size or data stream size.

GetFileObject(*data_stream_name=''*)

Retrieves a file-like object of a specific data stream.

Parameters
data_stream_name (*Optional[str]*) – data stream name, where an empty string represents the default data stream.

Returns
file-like object or None.

Return type
TSKFileIO

GetLinkedFileEntry()

Retrieves the linked file entry, e.g. for a symbolic link.

Returns
linked file entry or None.

Return type
TSKFileEntry

`GetParentFileEntry()`

Retrieves the parent file entry.

Returns
parent file entry or None.

Return type
TSKFileEntry

`GetTSKFile()`

Retrieves the SleuthKit file object.

Returns
TSK file.

Return type
pytsk3.File

Raises
PathSpecError – if the path specification is missing inode and location.

`IsAllocated()`

Determines if the file entry is allocated.

Returns
True if the file entry is allocated.

Return type
bool

`TYPE_INDICATOR = 'TSK'`

`property access_time`

access time or None if not available.

Type
dfdatetime.DateTimeValues

`property backup_time`

backup time or None if not available.

Type
dfdatetime.DateTimeValues

`property change_time`

change time or None if not available.

Type
dfdatetime.DateTimeValues

`property creation_time`

creation time or None if not available.

Type
dfdatetime.DateTimeValues

`property deletion_time`

deletion time or None if not available.

Type
dfdatetime.DateTimeValues

property modification_time

modification time or None if not available.

Type
dfdatetime.DateTimeValues

property name

name of the file entry, which does not include the full path.

Raises
BackEndError – if pytsk3 returns a non UTF-8 formatted name.

Type
str

property size

size of the file entry in bytes or None if not available.

Type
int

class dfvfs.vfs.tsk_file_entry.TSKTime(**args: Any*, ***kwargs: Any*)

Bases: DateTimeValues

SleuthKit timestamp.

fraction_of_second

fraction of second, which is an integer that contains the number 100 nano seconds before Sleuthkit 4.2.0 or number of nano seconds in Sleuthkit 4.2.0 and later.

Type
int

CopyFromDateTimeString(*time_string*)

Copies a SleuthKit timestamp from a date and time string.

Parameters
time_string (*str*) – date and time value formatted as: YYYY-MM-DD hh:mm:ss.######[+-]##:##

Where # are numeric digits ranging from 0 to 9 and the seconds fraction can be either 3 or 6 digits. The time of day, seconds fraction and time zone offset are optional. The default time zone is UTC.

CopyToDateTimeString()

Copies the date time value to a date and time string.

Returns

date and time value formatted as:
YYYY-MM-DD hh:mm:ss or YYYY-MM-DD hh:mm:ss.####### or YYYY-MM-DD hh:mm:ss.#########

Return type
str

GetDate()

Retrieves the date represented by the date and time values.

Returns

year, month, day of month or (None, None, None)
if the date and time values do not represent a date.

Return type
tuple[int, int, int]

property timestamp

POSIX timestamp in microseconds or None if timestamp is not set.

Type
int

dfvfs.vfs.tsk_file_system module

The SleuthKit (TSK) file system implementation.

class dfvfs.vfs.tsk_file_system.**TSKFileSystem**(*resolver_context*, *path_spec*)

Bases: *FileSystem*

File system that uses pytsk3.

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the file entry exists.

Return type
bool

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
a file entry or None if not available.

Return type
TSKFileEntry

GetFsInfo()

Retrieves the file system info.

Returns
file system info.

Return type
pytsk3.FS_Info

GetFsType()

Retrieves the file system type.

Returns
file system type.

Return type
pytsk3.TSK_FS_TYPE_ENUM

`GetRootFileEntry()`

Retrieves the root file entry.

Returns
a file entry.

Return type
TSKFileEntry

`GetRootInode()`

Retrieves the root inode.

Returns
inode number or None if not available.

Return type
int

`GetTSKFileByPathSpec(path_spec)`

Retrieves the SleuthKit file object for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
TSK file.

Return type
pytsk3.File

Raises
PathSpecError – if the path specification is missing inode and location.

`IsExt()`

Determines if the file system is ext2, ext3 or ext4.

Returns
True if the file system is ext.

Return type
bool

`IsHFS()`

Determines if the file system is HFS, HFS+ or HFSX.

Returns
True if the file system is HFS.

Return type
bool

`IsNTFS()`

Determines if the file system is NTFS.

Returns
True if the file system is NTFS.

Return type
bool

TYPE_INDICATOR = 'TSK'

dfvfs.vfs.tsk_partition_directory module

The SleuthKit (TSK) partition directory implementation.

class dfvfs.vfs.tsk_partition_directory.TSKPartitionDirectory(*file_system*, *path_spec*)

Bases: *Directory*

File system directory that uses pytsk3.

dfvfs.vfs.tsk_partition_file_entry module

The SleuthKit (TSK) partition file entry implementation.

class dfvfs.vfs.tsk_partition_file_entry.TSKPartitionFileEntry(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*, *tsk_vs_part=None*)

Bases: *FileEntry*

File system file entry that uses pytsk3.

GetParentFileEntry()

Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
TSKPartitionFileEntry

GetTSKVsPart()

Retrieves the TSK volume system part.

Returns

a TSK volume system part or None if not
available.

Return type
pytsk3.TSK_VS_PART_INFO

IsAllocated()

Determines if the file entry is allocated.

Returns
True if the file entry is allocated.

Return type
bool

TYPE_INDICATOR = 'TSK_PARTITION'

property name

name of the file entry, which does not include the full path.

Type
str

property size
size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.tsk_partition_file_system module

The SleuthKit (TSK) partition file system implementation.

class dfvfs.vfs.tsk_partition_file_system.**TSKPartitionFileSystem**(*resolver_context*, *path_spec*)
Bases: *FileSystem*

Class that implements a file system object using pytsk3.

FileEntryExistsByPathSpec(*path_spec*)
Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – a path specification.

Returns
True if the file entry exists or false otherwise.

Return type
bool

GetFileEntryByPathSpec(*path_spec*)
Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
a file entry or None of not available.

Return type
TSKPartitionFileEntry

GetRootFileEntry()
Retrieves the root file entry.

Returns
a file entry or None of not available.

Return type
TSKPartitionFileEntry

GetTSKVolume()
Retrieves the TSK volume object.

Returns
a TSK volume object.

Return type
pytsk3.Volume_Info

TYPE_INDICATOR = 'TSK_PARTITION'

dfvfs.vfs.vshadow_directory module

The Volume Shadow Snapshots (VSS) directory implementation.

class dfvfs.vfs.vshadow_directory.**VShadowDirectory**(*file_system*, *path_spec*)

Bases: *Directory*

File system directory that uses pyvshadow.

dfvfs.vfs.vshadow_file_entry module

The Volume Shadow Snapshots (VSS) file entry implementation.

class dfvfs.vfs.vshadow_file_entry.**VShadowFileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*)

Bases: *FileEntry*

File system file entry that uses pyvshadow.

GetParentFileEntry()

Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
FileEntry

GetVShadowStore()

Retrieves a VSS store.

Returns
a VSS store or None if not available.

Return type
pyvshadow.store

HasExternalData()

Determines if the file entry has external stored data.

Returns
True if the file entry has external data.

Return type
bool

TYPE_INDICATOR = 'VSHADOW'

property creation_time

creation time or None if not available.

Type
dfdatetime.DateTimeValues

property name

name of the file entry, which does not include the full path.

Type
str

property size
size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.vshadow_file_system module

The Volume Shadow Snapshots (VSS) file system implementation.

class dfvfs.vfs.vshadow_file_system.**VShadowFileSystem**(*resolver_context*, *path_spec*)

Bases: *FileSystem*

File system that uses pyvshadow.

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the file entry exists.

Return type
bool

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
file entry or None if not available.

Return type
VShadowFileEntry

GetRootFileEntry()

Retrieves the root file entry.

Returns
file entry or None if not available.

Return type
VShadowFileEntry

GetVShadowStoreByPathSpec(*path_spec*)

Retrieves a VSS store for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
a VSS store or None if not available.

Return type
pyvshadow.store

GetVShadowVolume()

Retrieves a VSS volume.

Returns
a VSS volume.

Return type
pyvshadow.volume

TYPE_INDICATOR = 'VSHADOW'

dfvfs.vfs.xfs_attribute module

The XFS attribute implementation.

class dfvfs.vfs.xfs_attribute.**XFSExtendedAttribute**(*fsxfs_extended_attribute*)

Bases: *Attribute*

XFS extended attribute that uses pyfsxfs.

GetExtents()

Retrieves the extents.

Returns
the extents of the attribute data.

Return type
list[*Extent*]

get_offset()

Retrieves the current offset into the file input/output (IO) object.

Returns
current offset into the file input/output (IO) object.

Return type
int

get_size()

Retrieves the size of the file input/output (IO) object.

Returns
size of the file input/output (IO) object.

Return type
int

property name

name.

Type
str

read(*size=None*)

Reads a byte string from the file input/output (IO) object.

The function will read a byte string of the specified size or all of the remaining data if no size was specified.

Parameters
size (*Optional[int]*) – number of bytes to read, where None is all remaining data.

Returns
data read.

Return type
bytes

Raises

- **IOError** – if the read failed.
- **OSError** – if the read failed.

seek(*offset*, *whence=0*)

Seeks to an offset within the file input/output (IO) object.

Parameters

- **offset** (*int*) – offset to seek.
- **whence** (*Optional[int]*) – value that indicates whether offset is an absolute or relative position within the file.

Raises

- **IOError** – if the seek failed.
- **OSError** – if the seek failed.

seekable()

Determines if a file input/output (IO) object is seekable.

Returns
True since a file IO object provides a seek method.

Return type
bool

tell()

Retrieves the current offset into the file input/output (IO) object.

dfvfs.vfs.xfs_directory module

The XFS directory implementation.

class dfvfs.vfs.xfs_directory.**XFSDirectory**(*file_system*, *path_spec*, *fsxfs_file_entry*)

Bases: *Directory*

File system directory that uses pyfsxfs.

dfvfs.vfs.xfs_file_entry module

The XFS file entry implementation.

class dfvfs.vfs.xfs_file_entry.**XFSFileEntry**(*resolver_context*, *file_system*, *path_spec*, *fsxfs_file_entry=None*, *is_root=False*, *is_virtual=False*)

Bases: *FileEntry*

File system file entry that uses pyfsxfs.

`GetExtents()`

Retrieves the extents.

Returns
the extents.

Return type
list[*Extent*]

`GetLinkedFileEntry()`

Retrieves the linked file entry, e.g. for a symbolic link.

Returns
linked file entry or None if not available.

Return type
XFSFileEntry

`GetParentFileEntry()`

Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
XFSFileEntry

`GetXFSFileEntry()`

Retrieves the XFS file entry.

Returns
XFS file entry.

Return type
pyfsxfs.file_entry

`TYPE_INDICATOR = 'XFS'`

`property access_time`

access time or None if not available.

Type
dfdatetime.DateTimeValues

`property change_time`

change time or None if not available.

Type
dfdatetime.DateTimeValues

`property creation_time`

creation time or None if not available.

Type
dfdatetime.DateTimeValues

`property modification_time`

modification time or None if not available.

Type
dfdatetime.DateTimeValues

property name

name of the file entry, which does not include the full path.

Type
str

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.xfs_file_system module

The XFS file system implementation.

class dfvfs.vfs.xfs_file_system.**XFSFileSystem**(*resolver_context*, *path_spec*)

Bases: *FileSystem*

File system that uses pyfsxfs.

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification.

Returns
True if the file entry exists.

Return type
bool

Raises
BackEndError – if the file entry cannot be opened.

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification.

Returns
file entry or None if not available.

Return type
XFSFileEntry

Raises
BackEndError – if the file entry cannot be opened.

GetRootFileEntry()

Retrieves the root file entry.

Returns
file entry.

Return type
XFSFileEntry

GetXFSFileEntryByPathSpec(*path_spec*)

Retrieves the XFS file entry for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
file entry.

Return type
pyfsxfs.file_entry

Raises
PathSpecError – if the path specification is missing location and inode.

TYPE_INDICATOR = 'XFS'

dfvfs.vfs.zip_directory module

The ZIP directory implementation.

class dfvfs.vfs.zip_directory.**ZIPDirectory**(*file_system*, *path_spec*)

Bases: *Directory*

File system directory that uses zipfile.

dfvfs.vfs.zip_file_entry module

The ZIP file entry implementation.

class dfvfs.vfs.zip_file_entry.**ZipFileEntry**(*resolver_context*, *file_system*, *path_spec*, *is_root=False*, *is_virtual=False*, *zip_info=None*)

Bases: *FileEntry*

File system file entry that uses zipfile.

GetParentFileEntry()

Retrieves the parent file entry.

Returns
parent file entry or None if not available.

Return type
ZipFileEntry

GetZipInfo()

Retrieves the ZIP info object.

Returns
a ZIP info object or None if not available.

Return type
zipfile.ZipInfo

Raises
PathSpecError – if the path specification is incorrect.

TYPE_INDICATOR = 'ZIP'

property modification_time

modification time or None if not available.

Type
dfdatetime.DateTimeValues

property name

name of the file entry, without the full path.

Type
str

property size

size of the file entry in bytes or None if not available.

Type
int

dfvfs.vfs.zip_file_system module

The ZIP file system implementation.

class dfvfs.vfs.zip_file_system.**ZipFileSystem**(*resolver_context*, *path_spec*, *encoding='utf-8'*)

Bases: *FileSystem*

File system that uses zipfile.

encoding

encoding of the file entry name.

Type
str

FileEntryExistsByPathSpec(*path_spec*)

Determines if a file entry for a path specification exists.

Parameters
path_spec (PathSpec) – path specification of the file entry.

Returns
True if the file entry exists.

Return type
bool

GetFileEntryByPathSpec(*path_spec*)

Retrieves a file entry for a path specification.

Parameters
path_spec (PathSpec) – path specification of the file entry.

Returns
a file entry or None.

Return type
ZipFileEntry

GetRootFileEntry()

Retrieves the root file entry.

Returns
a file entry or None.

Return type
ZipFileEntry

`GetZipFile()`

Retrieves the ZIP file object.

Returns
a ZIP file object or None.

Return type
zipfile.ZipFile

`GetZipInfoByPathSpec(path_spec)`

Retrieves the ZIP info for a path specification.

Parameters
path_spec (PathSpec) – a path specification.

Returns
a ZIP info object or None if not available.

Return type
zipfile.ZipInfo

Raises
PathSpecError – if the path specification is incorrect.

`TYPE_INDICATOR = 'ZIP'`

Module contents

6.1.15 dfvfs.volume package

Submodules

dfvfs.volume.apfs_volume_system module

The Apple File System (APFS) volume system.

`class dfvfs.volume.apfs_volume_system.APFSVolume(file_entry)`

Bases: *Volume*

Volume that uses pyfsapfs.

`class dfvfs.volume.apfs_volume_system.APFSVolumeSystem`

Bases: *VolumeSystem*

Volume system that uses pyfsapfs.

`TYPE_INDICATOR = 'APFS_CONTAINER'`

`VOLUME_IDENTIFIER_PREFIX = 'apfs'`

dfvfs.volume.cs_volume_system module

The Core Storage (CS) volume system.

class dfvfs.volume.cs_volume_system.**CSVolume**(*file_entry*)

Bases: *Volume*

Volume that uses pyfvde.

class dfvfs.volume.cs_volume_system.**CSVolumeSystem**

Bases: *VolumeSystem*

Volume system that uses pyfvde.

TYPE_INDICATOR = 'CS'

VOLUME_IDENTIFIER_PREFIX = 'cs'

dfvfs.volume.factory module

The volume system factory.

class dfvfs.volume.factory.**Factory**

Bases: object

Volume system factory.

classmethod DeregisterVolumeSystem(*volume_system_type*)

Deregisters a path specification type.

Parameters
volume_system_type (*type*) – path specification type.

Raises
KeyError – if path specification type is not registered.

classmethod NewVolumeSystem(*type_indicator*)

Creates a new path specification for the specific type indicator.

Parameters
type_indicator (*str*) – type indicator.

Returns
path specification.

Return type
VolumeSystem

Raises
KeyError – if path specification is not registered.

classmethod RegisterVolumeSystem(*volume_system_type*)

Registers a path specification type.

Parameters
volume_system_type (*type*) – path specification type.

Raises
KeyError – if path specification type is already registered.

dfvfs.volume.gpt_volume_system module

The GUID Partition Table (GPT) volume system.

class dfvfs.volume.gpt_volume_system.**GPTVolume**(*file_entry*)

Bases: *Volume*

Volume that uses pyvsgpt.

class dfvfs.volume.gpt_volume_system.**GPTVolumeSystem**

Bases: *VolumeSystem*

Volume system that uses pyvsgpt.

TYPE_INDICATOR = 'GPT'

VOLUME_IDENTIFIER_PREFIX = 'p'

dfvfs.volume.lvm_volume_system module

The Logical Volume Manager (LVM) volume system.

class dfvfs.volume.lvm_volume_system.**LVMVolume**(*file_entry*)

Bases: *Volume*

Volume that uses pyvslvm.

class dfvfs.volume.lvm_volume_system.**LVMVolumeSystem**

Bases: *VolumeSystem*

Volume system that uses pyvslvm.

TYPE_INDICATOR = 'LVM'

VOLUME_IDENTIFIER_PREFIX = 'lvm'

dfvfs.volume.tsk_volume_system module

The SleuthKit (TSK) volume system.

class dfvfs.volume.tsk_volume_system.**TSKVolume**(*file_entry*, *bytes_per_sector*)

Bases: *Volume*

Volume that uses pytsk3.

class dfvfs.volume.tsk_volume_system.**TSKVolumeSystem**

Bases: *VolumeSystem*

Volume system that uses pytsk3.

TYPE_INDICATOR = 'TSK_PARTITION'

VOLUME_IDENTIFIER_PREFIX = 'p'

dfvfs.volume.volume_system module

The Virtual File System (VFS) volume system interface.

class dfvfs.volume.volume_system.**Volume**(*identifier*)

Bases: object

The VFS volume interface.

GetAttribute(*identifier*)

Retrieves a specific attribute.

Parameters
identifier (*str*) – identifier of the attribute within the volume.

Returns
volume attribute or None if not available.

Return type
VolumeAttribute

HasExternalData()

Determines if the volume has external stored data.

Returns
True if the volume has external stored data.

Return type
bool

property attributes

volume attributes generator.

Type
generator[*VolumeAttribute*]

property extents

volume extents.

Type
list[*VolumeExtent*]

property number_of_attributes

number of attributes.

Type
int

property number_of_extents

number of extents.

Type
int

class dfvfs.volume.volume_system.**VolumeAttribute**(*identifier*, *value*)

Bases: object

The VFS volume attribute.

class dfvfs.volume.volume_system.**VolumeExtent**(*offset*, *size*, *extent_type=0*)

Bases: object

The VFS volume extent.

EXTENT_TYPE_DATA = 0

EXTENT_TYPE_SPARSE = 1

class dfvfs.volume.volume_system.**VolumeSystem**

Bases: object

The VFS volume system interface.

GetSectionByIndex(*section_index*)

Retrieves a specific section based on the index.

Parameters
section_index (*int*) – index of the section.

Returns
a volume extent or None if not available.

Return type
VolumeExtent

GetVolumeByIdentifier(*volume_identifier*)

Retrieves a specific volume based on the identifier.

Parameters
volume_identifier (*str*) – identifier of the volume within the volume system.

Returns
a volume.

Return type
Volume

GetVolumeByIndex(*volume_index*)

Retrieves a specific volume based on the index.

Parameters
volume_index (*int*) – index of the volume.

Returns
a volume or None if not available.

Return type
Volume

Open(*path_spec*)

Opens a volume defined by path specification.

Parameters
path_spec (PathSpec) – a path specification.

Raises
VolumeSystemError – if the virtual file system representing the volume system could not be resolved.

TYPE_INDICATOR = None

VOLUME_IDENTIFIER_PREFIX = 'v'

property number_of_sections

number of sections.

Type
int

property number_of_volumes

number of volumes.

Type
int

property sections

sections.

Type
list[*VolumeExtent*]

property volume_identifiers

volume identifiers.

Type
list[str]

property volumes

volumes generator.

Type
generator(*Volume*)

dfvfs.volume.vshadow_volume_system module

The Volume Shadow Snapshots (VSS) volume system.

class dfvfs.volume.vshadow_volume_system.**VShadowVolume**(*file_entry*)

Bases: *Volume*

Volume that uses pyvshadow.

HasExternalData()

Determines if the volume has external stored data.

Returns
True if the volume has external stored data.

Return type
bool

class dfvfs.volume.vshadow_volume_system.**VShadowVolumeSystem**

Bases: *VolumeSystem*

Volume system that uses pyvshadow.

TYPE_INDICATOR = 'VSHADOW'

VOLUME_IDENTIFIER_PREFIX = 'vss'

Module contents

Imports for the volume system factory.

6.2 Module contents

Digital Forensics Virtual File System (dfVFS).

dfVFS, or Digital Forensics Virtual File System, is a Python module that provides read-only access to file-system objects from various storage media types and file formats.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

d

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z