
dfVFS

Release 20221224

unknown

Dec 29, 2022

CONTENTS

1	Getting started	3
1.1	Installation instructions	3
2	Code snippets	5
2.1	Working with storage media images and devices	5
3	Path specifications	9
3.1	Terminology	9
3.2	Type indicators	9
3.3	Addressing attributes	9
4	Developer documentation	15
4.1	Introduction	15
4.2	Helpers	16
4.3	Adding a new VFS type	19
4.4	Testing	23
5	Supported Formats	25
5.1	Storage media types	25
5.2	Volume systems	25
5.3	File systems	26
5.4	Compressed stream file types	26
5.5	Encoded stream file types	26
5.6	Encrypted stream file types	26
5.7	Archive file types	27
5.8	Other file types	27
6	dfvfs package	29
6.1	Subpackages	29
6.2	Module contents	259
7	Indices and tables	261
	Python Module Index	263
	Index	267

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](#).

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
```


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}')
```


PATH SPECIFICATIONS

3.1 Terminology

In dfVFS a path specification 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 as 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](#).

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 node 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
```

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: [libfsxt/pyfsxt](#) 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

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 (regex) 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.VShadowAnalyzerHelperBases: *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.XFSAnalyzerHelperBases: *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 exist.

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.CSCredentialsBases: *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.EncryptedStreamCredentialsBases: *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 exist.

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 exist.

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 pyfssect.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 node 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 exist 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 exist.

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 exist 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 function 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 exist, 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 exist, 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.APFSToContainerPathSpecGetVolumeIndex(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.APFSToContainerPathSpecUnlockVolume(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_\(programming\)](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.zipfile module

Gzip compressed stream file.

class `dfvfs.lib.zipfile.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 exist.

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',
'encryption_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 exist.

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*LUKSDEFfile***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*LUKSDEFfileSystem***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.NTFSResolverHelperBases: *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.OSResolverHelperBases: *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.TSKPartitionResolverHelperBases: *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.TSKResolverHelperBases: *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.XFSResolverHelperBases: *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.ZipResolverHelperBases: *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 typelist[*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.APFSEntry*(*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

GetFVDEVVolumeGroup()

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.

Typegenerator[*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`(*fs_ext_extended_attribute*)

Bases: *Attribute*

EXT extended attribute that uses pyfsex.

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 pyfsxt.

dfvfs.vfs.ext_file_entry module

The EXT file entry implementation.

```
class dfvfs.vfs.ext_file_entry.EXTFileEntry(resolver_context, file_system, path_spec,
                                             fsxt_file_entry=None, is_root=False, is_virtual=False)
```

Bases: *FileEntry*

File system file entry that uses pyfsxt.

GetEXTFileEntry()

Retrieves the EXT file entry.

Returns

EXT file entry.

Return type

pyfsxt.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 pyfsxt.

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 typedict[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 typelist[`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

GetLUKSDEVVolume()

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

`pyvslm.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.NTFSEntry(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

pyfwnet.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 typelist[*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 typelist[*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.

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

d

dfvfs, 259
dfvfs.analyzer, 45
dfvfs.analyzer.analyzer, 29
dfvfs.analyzer.analyzer_helper, 31
dfvfs.analyzer.apfs_analyzer_helper, 32
dfvfs.analyzer.apfs_container_analyzer_helper, 32
dfvfs.analyzer.bde_analyzer_helper, 32
dfvfs.analyzer.bzip2_analyzer_helper, 33
dfvfs.analyzer.cpio_analyzer_helper, 33
dfvfs.analyzer.cs_analyzer_helper, 34
dfvfs.analyzer.ewf_analyzer_helper, 34
dfvfs.analyzer.ext_analyzer_helper, 34
dfvfs.analyzer.fat_analyzer_helper, 35
dfvfs.analyzer.gpt_analyzer_helper, 36
dfvfs.analyzer.gzip_analyzer_helper, 36
dfvfs.analyzer.hfs_analyzer_helper, 37
dfvfs.analyzer.luksde_analyzer_helper, 37
dfvfs.analyzer.lvm_analyzer_helper, 38
dfvfs.analyzer.modi_analyzer_helper, 38
dfvfs.analyzer.ntfs_analyzer_helper, 38
dfvfs.analyzer.phdi_analyzer_helper, 39
dfvfs.analyzer.qcow_analyzer_helper, 39
dfvfs.analyzer.specification, 40
dfvfs.analyzer.tar_analyzer_helper, 42
dfvfs.analyzer.tsk_analyzer_helper, 42
dfvfs.analyzer.tsk_partition_analyzer_helper, 42
dfvfs.analyzer.vhdi_analyzer_helper, 43
dfvfs.analyzer.vmdk_analyzer_helper, 43
dfvfs.analyzer.vshadow_analyzer_helper, 44
dfvfs.analyzer.xfs_analyzer_helper, 44
dfvfs.analyzer.xz_analyzer_helper, 45
dfvfs.analyzer.zip_analyzer_helper, 45
dfvfs.compression, 49
dfvfs.compression.bzip2_decompressor, 46
dfvfs.compression.decompressor, 46
dfvfs.compression.manager, 47
dfvfs.compression.xz_decompressor, 48
dfvfs.compression.zlib_decompressor, 48
dfvfs.credentials, 52
dfvfs.credentials.apfs_credentials, 49
dfvfs.credentials.bde_credentials, 49
dfvfs.credentials.credentials, 49
dfvfs.credentials.cs_credentials, 50
dfvfs.credentials.encrypted_stream_credentials, 50
dfvfs.credentials.keychain, 50
dfvfs.credentials.luksde_credentials, 51
dfvfs.credentials.manager, 52
dfvfs.encoding, 55
dfvfs.encoding.base16_decoder, 53
dfvfs.encoding.base32_decoder, 53
dfvfs.encoding.base64_decoder, 54
dfvfs.encoding.decoder, 54
dfvfs.encoding.manager, 54
dfvfs.encryption, 58
dfvfs.encryption.aes_decrypter, 55
dfvfs.encryption.blowfish_decrypter, 56
dfvfs.encryption.decrypter, 56
dfvfs.encryption.des3_decrypter, 57
dfvfs.encryption.manager, 57
dfvfs.encryption.rc4_decrypter, 58
dfvfs.file_io, 92
dfvfs.file_io.apfs_file_io, 58
dfvfs.file_io.bde_file_io, 60
dfvfs.file_io.compressed_stream_io, 60
dfvfs.file_io.cpio_file_io, 61
dfvfs.file_io.cs_file_io, 62
dfvfs.file_io.data_range_io, 64
dfvfs.file_io.encoded_stream_io, 65
dfvfs.file_io.encrypted_stream_io, 66
dfvfs.file_io.ewf_file_io, 68
dfvfs.file_io.ext_file_io, 68
dfvfs.file_io.fake_file_io, 69
dfvfs.file_io.fat_file_io, 70
dfvfs.file_io.file_io, 72
dfvfs.file_io.file_object_io, 73
dfvfs.file_io.gpt_file_io, 75
dfvfs.file_io.gzip_file_io, 76
dfvfs.file_io.hfs_file_io, 77
dfvfs.file_io.luksde_file_io, 78
dfvfs.file_io.lvm_file_io, 78

- dfvfs.file_io.modi_file_io, 79
- dfvfs.file_io.ntfs_file_io, 80
- dfvfs.file_io.os_file_io, 81
- dfvfs.file_io.phdi_file_io, 82
- dfvfs.file_io.qcow_file_io, 82
- dfvfs.file_io.raw_file_io, 83
- dfvfs.file_io.sqlite_blob_file_io, 83
- dfvfs.file_io.tar_file_io, 85
- dfvfs.file_io.tsk_file_io, 86
- dfvfs.file_io.tsk_partition_file_io, 87
- dfvfs.file_io.vhdi_file_io, 87
- dfvfs.file_io.vmdk_file_io, 88
- dfvfs.file_io.vshadow_file_io, 88
- dfvfs.file_io.xfs_file_io, 89
- dfvfs.file_io.zip_file_io, 90
- dfvfs.helpers, 113
 - dfvfs.helpers.command_line, 92
 - dfvfs.helpers.data_slice, 95
 - dfvfs.helpers.fake_file_system_builder, 96
 - dfvfs.helpers.file_system_searcher, 97
 - dfvfs.helpers.source_scanner, 100
 - dfvfs.helpers.text_file, 107
 - dfvfs.helpers.volume_scanner, 108
 - dfvfs.helpers.windows_path_resolver, 112
- dfvfs.lib, 127
 - dfvfs.lib.apfs_helper, 113
 - dfvfs.lib.bde_helper, 113
 - dfvfs.lib.cpio, 114
 - dfvfs.lib.cs_helper, 116
 - dfvfs.lib.data_format, 117
 - dfvfs.lib.decorators, 117
 - dfvfs.lib.definitions, 117
 - dfvfs.lib.errors, 117
 - dfvfs.lib.ewf_helper, 118
 - dfvfs.lib.glob2regex, 119
 - dfvfs.lib.gpt_helper, 119
 - dfvfs.lib.gzipfile, 119
 - dfvfs.lib.luksde_helper, 122
 - dfvfs.lib.lvm_helper, 123
 - dfvfs.lib.raw_helper, 123
 - dfvfs.lib.sqlite_database, 123
 - dfvfs.lib.tsk_image, 125
 - dfvfs.lib.tsk_partition, 126
 - dfvfs.lib.vshadow_helper, 127
- dfvfs.mount, 128
 - dfvfs.mount.manager, 127
- dfvfs.path, 146
 - dfvfs.path.apfs_container_path_spec, 128
 - dfvfs.path.apfs_path_spec, 129
 - dfvfs.path.bde_path_spec, 129
 - dfvfs.path.compressed_stream_path_spec, 130
 - dfvfs.path.cpio_path_spec, 130
 - dfvfs.path.cs_path_spec, 131
 - dfvfs.path.data_range_path_spec, 131
 - dfvfs.path.encoded_stream_path_spec, 132
 - dfvfs.path.encrypted_stream_path_spec, 132
 - dfvfs.path.ewf_path_spec, 133
 - dfvfs.path.ext_path_spec, 133
 - dfvfs.path.factory, 134
 - dfvfs.path.fake_path_spec, 135
 - dfvfs.path.fat_path_spec, 135
 - dfvfs.path.gpt_path_spec, 136
 - dfvfs.path.gzip_path_spec, 136
 - dfvfs.path.hfs_path_spec, 137
 - dfvfs.path.location_path_spec, 137
 - dfvfs.path.luksde_path_spec, 138
 - dfvfs.path.lvm_path_spec, 138
 - dfvfs.path.modi_path_spec, 139
 - dfvfs.path.mount_path_spec, 139
 - dfvfs.path.ntfs_path_spec, 139
 - dfvfs.path.os_path_spec, 140
 - dfvfs.path.path_spec, 140
 - dfvfs.path.phdi_path_spec, 142
 - dfvfs.path.qcow_path_spec, 142
 - dfvfs.path.raw_path_spec, 142
 - dfvfs.path.sqlite_blob_path_spec, 142
 - dfvfs.path.tar_path_spec, 143
 - dfvfs.path.tsk_partition_path_spec, 143
 - dfvfs.path.tsk_path_spec, 144
 - dfvfs.path.vhdi_path_spec, 145
 - dfvfs.path.vmdk_path_spec, 145
 - dfvfs.path.vshadow_path_spec, 145
 - dfvfs.path.xfs_path_spec, 146
 - dfvfs.path.zip_path_spec, 146
- dfvfs.resolver, 149
 - dfvfs.resolver.context, 146
 - dfvfs.resolver.resolver, 148
- dfvfs.resolver_helpers, 171
 - dfvfs.resolver_helpers.apfs_container_resolver_helper, 149
 - dfvfs.resolver_helpers.apfs_resolver_helper, 150
 - dfvfs.resolver_helpers.bde_resolver_helper, 151
 - dfvfs.resolver_helpers.compressed_stream_resolver_helper, 151
 - dfvfs.resolver_helpers.cpio_resolver_helper, 152
 - dfvfs.resolver_helpers.cs_resolver_helper, 153
 - dfvfs.resolver_helpers.data_range_resolver_helper, 153
 - dfvfs.resolver_helpers.encoded_stream_resolver_helper, 154
 - dfvfs.resolver_helpers.encrypted_stream_resolver_helper, 155
 - dfvfs.resolver_helpers.ewf_resolver_helper, 155

dfvfs.resolver_helpers.ext_resolver_helper, 156
 dfvfs.resolver_helpers.fake_resolver_helper, 156
 dfvfs.resolver_helpers.fat_resolver_helper, 157
 dfvfs.resolver_helpers.gpt_resolver_helper, 157
 dfvfs.resolver_helpers.gzip_resolver_helper, 158
 dfvfs.resolver_helpers.hfs_resolver_helper, 159
 dfvfs.resolver_helpers.luksde_resolver_helper, 159
 dfvfs.resolver_helpers.lvm_resolver_helper, 160
 dfvfs.resolver_helpers.manager, 161
 dfvfs.resolver_helpers.modi_resolver_helper, 161
 dfvfs.resolver_helpers.ntfs_resolver_helper, 162
 dfvfs.resolver_helpers.os_resolver_helper, 162
 dfvfs.resolver_helpers.phdi_resolver_helper, 163
 dfvfs.resolver_helpers.qcow_resolver_helper, 164
 dfvfs.resolver_helpers.raw_resolver_helper, 164
 dfvfs.resolver_helpers.resolver_helper, 165
 dfvfs.resolver_helpers.sqlite_blob_resolver_helper, 165
 dfvfs.resolver_helpers.tar_resolver_helper, 166
 dfvfs.resolver_helpers.tsk_partition_resolver_helper, 167
 dfvfs.resolver_helpers.tsk_resolver_helper, 167
 dfvfs.resolver_helpers.vhdi_resolver_helper, 168
 dfvfs.resolver_helpers.vmdk_resolver_helper, 169
 dfvfs.resolver_helpers.vshadow_resolver_helper, 169
 dfvfs.resolver_helpers.xfs_resolver_helper, 170
 dfvfs.resolver_helpers.zip_resolver_helper, 170
 dfvfs.serializer, 172
 dfvfs.serializer.json_serializer, 171
 dfvfs.serializer.serializer, 172
 dfvfs.vfs, 253
 dfvfs.vfs.apfs_attribute, 172
 dfvfs.vfs.apfs_container_directory, 174
 dfvfs.vfs.apfs_container_file_entry, 174
 dfvfs.vfs.apfs_container_file_system, 175
 dfvfs.vfs.apfs_directory, 176
 dfvfs.vfs.apfs_file_entry, 176
 dfvfs.vfs.apfs_file_system, 178
 dfvfs.vfs.attribute, 179
 dfvfs.vfs.bde_file_entry, 181
 dfvfs.vfs.bde_file_system, 181
 dfvfs.vfs.compressed_stream_file_entry, 182
 dfvfs.vfs.compressed_stream_file_system, 182
 dfvfs.vfs.cpio_directory, 183
 dfvfs.vfs.cpio_file_entry, 183
 dfvfs.vfs.cpio_file_system, 184
 dfvfs.vfs.cs_directory, 185
 dfvfs.vfs.cs_file_entry, 185
 dfvfs.vfs.cs_file_system, 186
 dfvfs.vfs.data_range_file_entry, 187
 dfvfs.vfs.data_range_file_system, 188
 dfvfs.vfs.data_stream, 188
 dfvfs.vfs.directory, 189
 dfvfs.vfs.encoded_stream_file_entry, 189
 dfvfs.vfs.encoded_stream_file_system, 189
 dfvfs.vfs.encrypted_stream_file_entry, 190
 dfvfs.vfs.encrypted_stream_file_system, 190
 dfvfs.vfs.ext_attribute, 191
 dfvfs.vfs.ext_directory, 192
 dfvfs.vfs.ext_file_entry, 193
 dfvfs.vfs.ext_file_system, 194
 dfvfs.vfs.extent, 195
 dfvfs.vfs.fake_directory, 196
 dfvfs.vfs.fake_file_entry, 196
 dfvfs.vfs.fake_file_system, 197
 dfvfs.vfs.fat_directory, 199
 dfvfs.vfs.fat_file_entry, 199
 dfvfs.vfs.fat_file_system, 200
 dfvfs.vfs.file_entry, 202
 dfvfs.vfs.file_system, 207
 dfvfs.vfs.gpt_directory, 210
 dfvfs.vfs.gpt_file_entry, 210
 dfvfs.vfs.gpt_file_system, 211
 dfvfs.vfs.gzip_file_entry, 212
 dfvfs.vfs.gzip_file_system, 212
 dfvfs.vfs.hfs_attribute, 213
 dfvfs.vfs.hfs_data_stream, 214
 dfvfs.vfs.hfs_directory, 215
 dfvfs.vfs.hfs_file_entry, 215
 dfvfs.vfs.hfs_file_system, 217
 dfvfs.vfs.luksde_file_entry, 218
 dfvfs.vfs.luksde_file_system, 218
 dfvfs.vfs.lvm_directory, 219
 dfvfs.vfs.lvm_file_entry, 219
 dfvfs.vfs.lvm_file_system, 220
 dfvfs.vfs.ntfs_attribute, 221
 dfvfs.vfs.ntfs_data_stream, 223

dfvfs.vfs.ntfs_directory, 224
dfvfs.vfs.ntfs_file_entry, 224
dfvfs.vfs.ntfs_file_system, 226
dfvfs.vfs.os_attribute, 227
dfvfs.vfs.os_directory, 228
dfvfs.vfs.os_file_entry, 229
dfvfs.vfs.os_file_system, 230
dfvfs.vfs.root_only_file_entry, 231
dfvfs.vfs.root_only_file_system, 231
dfvfs.vfs.sqlite_blob_directory, 232
dfvfs.vfs.sqlite_blob_file_entry, 232
dfvfs.vfs.sqlite_blob_file_system, 233
dfvfs.vfs.tar_directory, 233
dfvfs.vfs.tar_file_entry, 234
dfvfs.vfs.tar_file_system, 235
dfvfs.vfs.tsk_attribute, 236
dfvfs.vfs.tsk_data_stream, 237
dfvfs.vfs.tsk_directory, 238
dfvfs.vfs.tsk_file_entry, 238
dfvfs.vfs.tsk_file_system, 241
dfvfs.vfs.tsk_partition_directory, 243
dfvfs.vfs.tsk_partition_file_entry, 243
dfvfs.vfs.tsk_partition_file_system, 244
dfvfs.vfs.vshadow_directory, 245
dfvfs.vfs.vshadow_file_entry, 245
dfvfs.vfs.vshadow_file_system, 246
dfvfs.vfs.xfs_attribute, 247
dfvfs.vfs.xfs_directory, 248
dfvfs.vfs.xfs_file_entry, 248
dfvfs.vfs.xfs_file_system, 250
dfvfs.vfs.zip_directory, 251
dfvfs.vfs.zip_file_entry, 251
dfvfs.vfs.zip_file_system, 252
dfvfs.volume, 259
dfvfs.volume.apfs_volume_system, 253
dfvfs.volume.cs_volume_system, 254
dfvfs.volume.factory, 254
dfvfs.volume.gpt_volume_system, 255
dfvfs.volume.lvm_volume_system, 255
dfvfs.volume.tsk_volume_system, 255
dfvfs.volume.volume_system, 256
dfvfs.volume.vshadow_volume_system, 258

Symbols

`__del__()` (*dfvfs.file_io.file_io.FileIO* method), 72
`__del__()` (*dfvfs.vfs.file_system.FileSystem* method), 210
`__enter__()` (*dfvfs.helpers.data_slice.DataSlice* method), 95
`__enter__()` (*dfvfs.helpers.text_file.TextFile* method), 107
`__eq__()` (*dfvfs.path.path_spec.PathSpec* method), 141
`__exit__()` (*dfvfs.helpers.data_slice.DataSlice* method), 95
`__exit__()` (*dfvfs.helpers.text_file.TextFile* method), 107
`__getitem__()` (*dfvfs.helpers.data_slice.DataSlice* method), 95
`__hash__()` (*dfvfs.path.path_spec.PathSpec* method), 141
`__iter__()` (*dfvfs.helpers.text_file.TextFile* method), 107
`__len__()` (*dfvfs.helpers.data_slice.DataSlice* method), 96

A

`access_time` (*dfvfs.vfs.apfs_file_entry.APFSFileEntry* property), 177
`access_time` (*dfvfs.vfs.ext_file_entry.EXTFileEntry* property), 193
`access_time` (*dfvfs.vfs.fake_file_entry.FakeFileEntry* property), 196
`access_time` (*dfvfs.vfs.fat_file_entry.FATFileEntry* property), 200
`access_time` (*dfvfs.vfs.file_entry.FileEntry* property), 205
`access_time` (*dfvfs.vfs.hfs_file_entry.HFSFileEntry* property), 216
`access_time` (*dfvfs.vfs.ntfs_attribute.FileNameNTFSAttribute* property), 221
`access_time` (*dfvfs.vfs.ntfs_attribute.StandardInformationNTFSAttribute* property), 222
`access_time` (*dfvfs.vfs.ntfs_file_entry.NTFSFileEntry* property), 225

`access_time` (*dfvfs.vfs.os_file_entry.OSFileEntry* property), 229
`access_time` (*dfvfs.vfs.tsk_file_entry.TSKFileEntry* property), 239
`access_time` (*dfvfs.vfs.xfs_file_entry.XFSFileEntry* property), 249
`AccessError`, 117
`AddDirectory()` (*dfvfs.helpers.fake_file_system_builder.FakeFileSystemBuilder* method), 96
`added_time` (*dfvfs.vfs.apfs_file_entry.APFSFileEntry* property), 177
`added_time` (*dfvfs.vfs.file_entry.FileEntry* property), 205
`added_time` (*dfvfs.vfs.hfs_file_entry.HFSFileEntry* property), 216
`AddFile()` (*dfvfs.helpers.fake_file_system_builder.FakeFileSystemBuilder* method), 96
`AddFileEntry()` (*dfvfs.vfs.fake_file_system.FakeFileSystem* method), 197
`AddFileReadData()` (*dfvfs.helpers.fake_file_system_builder.FakeFileSystemBuilder* method), 96
`AddNewSignature()` (*dfvfs.analyzer.specification.FormatSpecification* method), 40
`AddNewSpecification()` (*dfvfs.analyzer.specification.FormatSpecificationStore* method), 40
`AddRow()` (*dfvfs.helpers.command_line.CLITabularTableView* method), 92
`AddScanNode()` (*dfvfs.helpers.source_scanner.SourceScannerContext* method), 103
`AddSpecification()` (*dfvfs.analyzer.specification.FormatSpecificationStore* method), 40
`AddSymbolicLink()` (*dfvfs.helpers.fake_file_system_builder.FakeFileSystemBuilder* method), 97
`AESDecrypter` (class in *dfvfs.encryption.aes_decrypter*), 55
`AnalyzeFileObject()` (*dfvfs.analyzer.analyzer_helper.AnalyzerHelper* method), 31
`AnalyzeFileObject()` (*dfvfs.analyzer.tsk_partition_analyzer_helper.TSKPartitionAnalyzer* method), 42
`Analyzer` (class in *dfvfs.analyzer.analyzer*), 29

AnalyzerHelper (class in *dfvfs.analyzer.analyzer_helper*), 31
 APFSAnalyzerHelper (class in *dfvfs.analyzer.apfs_analyzer_helper*), 32
 APFSContainerAnalyzerHelper (class in *dfvfs.analyzer.apfs_container_analyzer_helper*), 32
 APFSContainerDirectory (class in *dfvfs.vfs.apfs_container_directory*), 174
 APFSContainerFileEntry (class in *dfvfs.vfs.apfs_container_file_entry*), 174
 APFSContainerFileSystem (class in *dfvfs.vfs.apfs_container_file_system*), 175
 APFSContainerPathSpec (class in *dfvfs.path.apfs_container_path_spec*), 128
 APFSContainerPathSpecGetVolumeIndex() (in module *dfvfs.lib.apfs_helper*), 113
 APFSContainerResolverHelper (class in *dfvfs.resolver_helpers.apfs_container_resolver_helper*), 149
 APFSCredentials (class in *dfvfs.credentials.apfs_credentials*), 49
 APFSDirectory (class in *dfvfs.vfs.apfs_directory*), 176
 APFSExtendedAttribute (class in *dfvfs.vfs.apfs_attribute*), 172
 APFSFile (class in *dfvfs.file_io.apfs_file_io*), 58
 APFSFileEntry (class in *dfvfs.vfs.apfs_file_entry*), 176
 APFSFileSystem (class in *dfvfs.vfs.apfs_file_system*), 178
 APFSPathSpec (class in *dfvfs.path.apfs_path_spec*), 129
 APFSResolverHelper (class in *dfvfs.resolver_helpers.apfs_resolver_helper*), 150
 APFSUnlockVolume() (in module *dfvfs.lib.apfs_helper*), 113
 APFSVolume (class in *dfvfs.volume.apfs_volume_system*), 253
 APFSVolumeSystem (class in *dfvfs.volume.apfs_volume_system*), 253
 AtLastLocationSegment() (*dfvfs.helpers.file_system_searcher.FindSpec* method), 98
 Attribute (class in *dfvfs.vfs.attribute*), 179
 attribute_type (*dfvfs.vfs.ntfs_attribute.NTFSAttribute* property), 222
 attribute_type (*dfvfs.vfs.tsk_attribute.TSKAttribute* property), 236
 attributes (*dfvfs.vfs.file_entry.FileEntry* property), 206
 attributes (*dfvfs.volume.volume_system.Volume* property), 256
B
 BackEndError, 117
 backup_time (*dfvfs.vfs.file_entry.FileEntry* property), 206
 backup_time (*dfvfs.vfs.hfs_file_entry.HFSFileEntry* property), 216
 backup_time (*dfvfs.vfs.tsk_file_entry.TSKFileEntry* property), 239
 Base16Decoder (class in *dfvfs.encoding.base16_decoder*), 53
 Base32Decoder (class in *dfvfs.encoding.base32_decoder*), 53
 Base64Decoder (class in *dfvfs.encoding.base64_decoder*), 54
 BasenamePath() (*dfvfs.vfs.file_system.FileSystem* method), 207
 BDEAnalyzerHelper (class in *dfvfs.analyzer.bde_analyzer_helper*), 32
 BDECredentials (class in *dfvfs.credentials.bde_credentials*), 49
 BDEFile (class in *dfvfs.file_io.bde_file_io*), 60
 BDEFileEntry (class in *dfvfs.vfs.bde_file_entry*), 181
 BDEFileSystem (class in *dfvfs.vfs.bde_file_system*), 181
 BDEOpenVolume() (in module *dfvfs.lib.bde_helper*), 113
 BDEPathSpec (class in *dfvfs.path.bde_path_spec*), 129
 BDEResolverHelper (class in *dfvfs.resolver_helpers.bde_resolver_helper*), 151
 BDEUnlockVolume() (in module *dfvfs.lib.bde_helper*), 113
 BlowfishDecrypter (class in *dfvfs.encryption.blowfish_decrypter*), 56
 BZIP2AnalyzerHelper (class in *dfvfs.analyzer.bzip2_analyzer_helper*), 33
 BZIP2Decompressor (class in *dfvfs.compression.bzip2_decompressor*), 46
C
 CacheFileObject() (*dfvfs.resolver.context.Context* method), 146
 CacheFileSystem() (*dfvfs.resolver.context.Context* method), 147
 CacheFullError, 117
 change_time (*dfvfs.vfs.apfs_file_entry.APFSFileEntry* property), 177
 change_time (*dfvfs.vfs.ext_file_entry.EXTFileEntry* property), 193
 change_time (*dfvfs.vfs.fake_file_entry.FakeFileEntry* property), 196
 change_time (*dfvfs.vfs.file_entry.FileEntry* property), 206
 change_time (*dfvfs.vfs.hfs_file_entry.HFSFileEntry* property), 216
 change_time (*dfvfs.vfs.ntfs_file_entry.NTFSFileEntry* property), 225

change_time (*dfvfs.vfs.os_file_entry.OSFileEntry* property), 229
 change_time (*dfvfs.vfs.tsk_file_entry.TSKFileEntry* property), 239
 change_time (*dfvfs.vfs.xfs_file_entry.XFSFileEntry* property), 249
 cipher_mode (*dfvfs.path.encrypted_stream_path_spec.EncryptedStreamPathSpec* attribute), 132
 CLIInputReader (class in *dfvfs.helpers.command_line*), 92
 CLIOutputWriter (class in *dfvfs.helpers.command_line*), 92
 CLITabularTableView (class in *dfvfs.helpers.command_line*), 92
 CLIVolumeScannerMediator (class in *dfvfs.helpers.command_line*), 92
 Close() (*dfvfs.lib.cpio.CPIOArchiveFile* method), 114
 close() (*dfvfs.lib.gzipfile.GzipCompressedStream* method), 120
 Close() (*dfvfs.lib.sqlite_database.SQLiteDatabaseFile* method), 123
 close() (*dfvfs.lib.tsk_image.TSKFileSystemImage* method), 125
 column_name (*dfvfs.path.sqlite_blob_path_spec.SQLiteBlobPathSpec* attribute), 142
 comment (*dfvfs.lib.gzipfile.GzipMember* attribute), 121
 comments (*dfvfs.file_io.gzip_file_io.GzipFile* property), 76
 comparable (*dfvfs.path.apfs_container_path_spec.APFSContainerPathSpec* property), 128
 comparable (*dfvfs.path.apfs_path_spec.APFSPathSpec* property), 129
 comparable (*dfvfs.path.bde_path_spec.BDEPathSpec* property), 130
 comparable (*dfvfs.path.compressed_stream_path_spec.CompressedStreamPathSpec* property), 130
 comparable (*dfvfs.path.cs_path_spec.CSPathSpec* property), 131
 comparable (*dfvfs.path.data_range_path_spec.DataRangePathSpec* property), 132
 comparable (*dfvfs.path.encoded_stream_path_spec.EncodedStreamPathSpec* property), 132
 comparable (*dfvfs.path.encrypted_stream_path_spec.EncryptedStreamPathSpec* property), 133
 comparable (*dfvfs.path.ext_path_spec.EXTPathSpec* property), 134
 comparable (*dfvfs.path.fat_path_spec.FATPathSpec* property), 136
 comparable (*dfvfs.path.gpt_path_spec.GPTPathSpec* property), 136
 comparable (*dfvfs.path.hfs_path_spec.HFSPathSpec* property), 137
 comparable (*dfvfs.path.location_path_spec.LocationPathSpec* property), 137
 comparable (*dfvfs.path.luksde_path_spec.LUKSDEPathSpec* property), 138
 comparable (*dfvfs.path.lvm_path_spec.LVMPathSpec* property), 138
 comparable (*dfvfs.path.mount_path_spec.MountPathSpec* property), 139
 comparable (*dfvfs.path.ntfs_path_spec.NTFSPathSpec* property), 140
 comparable (*dfvfs.path.path_spec.PathSpec* property), 141
 comparable (*dfvfs.path.sqlite_blob_path_spec.SQLiteBlobPathSpec* property), 143
 comparable (*dfvfs.path.tsk_partition_path_spec.TSKPartitionPathSpec* property), 144
 comparable (*dfvfs.path.tsk_path_spec.TSKPathSpec* property), 144
 comparable (*dfvfs.path.vshadow_path_spec.VShadowPathSpec* property), 145
 comparable (*dfvfs.path.xfs_path_spec.XFSPathSpec* property), 146
 CompareLocation() (*dfvfs.helpers.file_system_searcher.FindSpec* method), 98
 CompareNameWithLocationSegment() (*dfvfs.helpers.file_system_searcher.FindSpec* method), 98
 CompareTraits() (*dfvfs.helpers.file_system_searcher.FindSpec* method), 99
 CompressedStream (class in *dfvfs.file_io.compressed_stream_io*), 60
 CompressedStreamFileEntry (class in *dfvfs.vfs.compressed_stream_file_entry*), 182
 CompressedStreamFileSystem (class in *dfvfs.vfs.compressed_stream_file_system*), 182
 CompressedStreamPathSpec (class in *dfvfs.path.compressed_stream_path_spec*), 130
 CompressedStreamResolverHelper (class in *dfvfs.resolver_helpers.compressed_stream_resolver_helper*), 130
 COMPRESSION_METHOD (*dfvfs.compression.bzip2_decompressor.BZIP2Decompressor* attribute), 46
 COMPRESSION_METHOD (*dfvfs.compression.xz_decompressor.LZMADecompressor* attribute), 48
 COMPRESSION_METHOD (*dfvfs.compression.xz_decompressor.XZDecompressor* attribute), 48
 COMPRESSION_METHOD (*dfvfs.compression.zlib_decompressor.DeflateDecompressor* attribute), 48
 COMPRESSION_METHOD (*dfvfs.compression.zlib_decompressor.ZlibDecompressor* attribute), 48
 compression_method (*dfvfs.path.compressed_stream_path_spec.CompressedStreamPathSpec* attribute), 130
 CompressionManager (class in *dfvfs.resolver_helpers.compressed_stream_resolver_helper*), 130

- dfvfs.compression.manager*), 47
- Context (class in *dfvfs.resolver.context*), 146
- CopyFromDateTimeString() (*dfvfs.vfs.tsk_file_entry.TSKTime* method), 240
- CopyToDateTimeString() (*dfvfs.vfs.tsk_file_entry.TSKTime* method), 240
- CopyToDict() (*dfvfs.path.path_spec.PathSpec* method), 140
- CPIOAnalyzerHelper (class in *dfvfs.analyzer.cpio_analyzer_helper*), 33
- CPIOArchiveFile (class in *dfvfs.lib.cpio*), 114
- CPIOArchiveFileEntry (class in *dfvfs.lib.cpio*), 115
- CPIODirectory (class in *dfvfs.vfs.cpio_directory*), 183
- CPIOFile (class in *dfvfs.file_io.cpio_file_io*), 61
- CPIOFileEntry (class in *dfvfs.vfs.cpio_file_entry*), 183
- CPIOFileSystem (class in *dfvfs.vfs.cpio_file_system*), 184
- CPIOPathSpec (class in *dfvfs.path.cpio_path_spec*), 130
- CPIOResolverHelper (class in *dfvfs.resolver_helpers.cpio_resolver_helper*), 152
- creation_time (*dfvfs.vfs.apfs_file_entry.APFSFileEntry* property), 177
- creation_time (*dfvfs.vfs.bde_file_entry.BDEFileEntry* property), 181
- creation_time (*dfvfs.vfs.ext_file_entry.EXTFileEntry* property), 193
- creation_time (*dfvfs.vfs.fat_file_entry.FATFileEntry* property), 200
- creation_time (*dfvfs.vfs.file_entry.FileEntry* property), 206
- creation_time (*dfvfs.vfs.hfs_file_entry.HFSFileEntry* property), 216
- creation_time (*dfvfs.vfs.ntfs_attribute.FileNameNTFSAttribute* property), 221
- creation_time (*dfvfs.vfs.ntfs_attribute.StandardInformationNTFSAttribute* property), 223
- creation_time (*dfvfs.vfs.ntfs_file_entry.NTFSFileEntry* property), 225
- creation_time (*dfvfs.vfs.os_file_entry.OSFileEntry* property), 229
- creation_time (*dfvfs.vfs.tsk_file_entry.TSKFileEntry* property), 239
- creation_time (*dfvfs.vfs.vshadow_file_entry.VShadowFileEntry* property), 245
- creation_time (*dfvfs.vfs.xfs_file_entry.XFSFileEntry* property), 249
- credential (*dfvfs.helpers.source_scanner.SourceScanNode* attribute), 100
- Credentials (class in *dfvfs.credentials.credentials*), 49
- CREDENTIALS (*dfvfs.credentials.apfs_credentials.APFSCredentials* attribute), 49
- CREDENTIALS (*dfvfs.credentials.bde_credentials.BDECredentials* attribute), 49
- CREDENTIALS (*dfvfs.credentials.cs_credentials.CSCredentials* attribute), 50
- CREDENTIALS (*dfvfs.credentials.encrypted_stream_credentials.EncryptedStreamCredentials* attribute), 50
- CREDENTIALS (*dfvfs.credentials.luksde_credentials.LUKSDECredentials* attribute), 51
- credentials (*dfvfs.helpers.volume_scanner.VolumeScannerOptions* attribute), 110
- CredentialsManager (class in *dfvfs.credentials.manager*), 52
- CryptographyBlockCipherDecrypter (class in *dfvfs.encryption.decrypter*), 56
- CSAnalyzerHelper (class in *dfvfs.analyzer.cs_analyzer_helper*), 34
- CSCredentials (class in *dfvfs.credentials.cs_credentials*), 50
- CSDirectory (class in *dfvfs.vfs.cs_directory*), 185
- CSFile (class in *dfvfs.file_io.cs_file_io*), 62
- CSFileEntry (class in *dfvfs.vfs.cs_file_entry*), 185
- CSFileSystem (class in *dfvfs.vfs.cs_file_system*), 186
- CSPathSpec (class in *dfvfs.path.cs_path_spec*), 131
- CSPathSpecGetVolumeIndex() (in module *dfvfs.lib.cs_helper*), 116
- CSResolverHelper (class in *dfvfs.resolver_helpers.cs_resolver_helper*), 153
- CSUnlockLogicalVolume() (in module *dfvfs.lib.cs_helper*), 116
- CSVolume (class in *dfvfs.volume.cs_volume_system*), 254
- CSVolumeSystem (class in *dfvfs.volume.cs_volume_system*), 254
- ## D
- data_offset (*dfvfs.lib.cpio.CPIOArchiveFileEntry* attribute), 115
- data_size (*dfvfs.lib.cpio.CPIOArchiveFileEntry* attribute), 115
- data_stream (*dfvfs.path.hfs_path_spec.HFSPathSpec* attribute), 137
- data_stream (*dfvfs.path.ntfs_path_spec.NTFSPathSpec* attribute), 139
- data_stream (*dfvfs.path.tsk_path_spec.TSKPathSpec* attribute), 144
- data_streams (*dfvfs.vfs.file_entry.FileEntry* property), 206
- DataFormat (class in *dfvfs.lib.data_format*), 117
- DataRange (class in *dfvfs.file_io.data_range_io*), 64
- DataRangeFileEntry (class in *dfvfs.vfs.data_range_file_entry*), 187
- DataRangeFileSystem (class in *dfvfs.vfs.data_range_file_system*), 188

DataRangePathSpec (class in *dfvfs.path.data_range_path_spec*), 131
 DataRangeResolverHelper (class in *dfvfs.resolver_helpers.data_range_resolver_helper*), 153
 DataSlice (class in *dfvfs.helpers.data_slice*), 95
 DataStream (class in *dfvfs.vfs.data_stream*), 188
 Decode() (*dfvfs.encoding.base16_decoder.Base16Decoder* method), 53
 Decode() (*dfvfs.encoding.base32_decoder.Base32Decoder* method), 53
 Decode() (*dfvfs.encoding.base64_decoder.Base64Decoder* method), 54
 Decode() (*dfvfs.encoding.decoder.Decoder* method), 54
 Decoder (class in *dfvfs.encoding.decoder*), 54
 Decompress() (*dfvfs.compression.bzip2_decompressor.BZip2Decompressor* method), 46
 Decompress() (*dfvfs.compression.decompressor.Decompressor* method), 46
 Decompress() (*dfvfs.compression.xz_decompressor.XZDecompressor* method), 48
 Decompress() (*dfvfs.compression.zlib_decompressor.ZlibDecompressor* method), 48
 Decompressor (class in *dfvfs.compression.decompressor*), 46
 Decrypt() (*dfvfs.encryption.decrypter.CryptographyBlockDecrypter* method), 56
 Decrypt() (*dfvfs.encryption.decrypter.Decrypter* method), 56
 Decrypt() (*dfvfs.encryption.rc4_decrypter.RC4Decrypter* method), 58
 Decrypter (class in *dfvfs.encryption.decrypter*), 56
 DeflateDecompressor (class in *dfvfs.compression.zlib_decompressor*), 48
 deletion_time (*dfvfs.vfs.ext_file_entry.EXTFileEntry* property), 194
 deletion_time (*dfvfs.vfs.file_entry.FileEntry* property), 206
 deletion_time (*dfvfs.vfs.tsk_file_entry.TSKFileEntry* property), 239
 deprecated() (in module *dfvfs.lib.decorators*), 117
 DeregisterCredentials() (*dfvfs.credentials.manager.CredentialsManager* class method), 52
 DeregisterDecoder() (*dfvfs.encoding.manager.EncodingManager* class method), 54
 DeregisterDecompressor() (*dfvfs.compression.manager.CompressionManager* class method), 47
 DeregisterDecrypter() (*dfvfs.encryption.manager.EncryptionManager* class method), 57
 DeregisterHelper() (*dfvfs.analyzer.analyzer.Analyzer* class method), 29
 DeregisterHelper() (*dfvfs.resolver_helpers.manager.ResolverHelperManager* class method), 161
 DeregisterMountPoint() (*dfvfs.mount.manager.MountPointManager* class method), 127
 DeregisterMountPoint() (*dfvfs.resolver.context.Context* method), 147
 DeregisterPathSpec() (*dfvfs.path.factory.Factory* class method), 134
 DeregisterVolumeSystem() (*dfvfs.volume.factory.Factory* class method), 254
 DES3Decrypter (class in *dfvfs.encryption.des3_decrypter*), 57
 DES3Decrypter (class in *dfvfs.vfs.attribute.StatAttribute* attribute), 179
 dfvfs module, 259
 dfvfs.analyzer module, 45
 dfvfs.analyzer.analyzer module, 29
 dfvfs.analyzer.analyzer_helper module, 31
 dfvfs.analyzer.apfs_analyzer_helper module, 32
 dfvfs.analyzer.apfs_container_analyzer_helper module, 32
 dfvfs.analyzer.bde_analyzer_helper module, 32
 dfvfs.analyzer.bzip2_analyzer_helper module, 33
 dfvfs.analyzer.cpio_analyzer_helper module, 33
 dfvfs.analyzer.cs_analyzer_helper module, 34
 dfvfs.analyzer.ewf_analyzer_helper module, 34
 dfvfs.analyzer.ext_analyzer_helper module, 34
 dfvfs.analyzer.fat_analyzer_helper module, 35
 dfvfs.analyzer.gpt_analyzer_helper module, 36
 dfvfs.analyzer.gzip_analyzer_helper module, 36
 dfvfs.analyzer.hfs_analyzer_helper module, 37
 dfvfs.analyzer.luksde_analyzer_helper module, 37
 dfvfs.analyzer.lvm_analyzer_helper module, 38
 dfvfs.analyzer.modi_analyzer_helper module, 38

module, 38
dfvfs.analyzer.ntfs_analyzer_helper
 module, 38
dfvfs.analyzer.phdi_analyzer_helper
 module, 39
dfvfs.analyzer.qcow_analyzer_helper
 module, 39
dfvfs.analyzer.specification
 module, 40
dfvfs.analyzer.tar_analyzer_helper
 module, 42
dfvfs.analyzer.tsk_analyzer_helper
 module, 42
dfvfs.analyzer.tsk_partition_analyzer_helper
 module, 42
dfvfs.analyzer.vhdi_analyzer_helper
 module, 43
dfvfs.analyzer.vmdk_analyzer_helper
 module, 43
dfvfs.analyzer.vshadow_analyzer_helper
 module, 44
dfvfs.analyzer.xfs_analyzer_helper
 module, 44
dfvfs.analyzer.xz_analyzer_helper
 module, 45
dfvfs.analyzer.zip_analyzer_helper
 module, 45
dfvfs.compression
 module, 49
dfvfs.compression.bzip2_decompressor
 module, 46
dfvfs.compression.decompressor
 module, 46
dfvfs.compression.manager
 module, 47
dfvfs.compression.xz_decompressor
 module, 48
dfvfs.compression.zlib_decompressor
 module, 48
dfvfs.credentials
 module, 52
dfvfs.credentials.apfs_credentials
 module, 49
dfvfs.credentials.bde_credentials
 module, 49
dfvfs.credentials.credentials
 module, 49
dfvfs.credentials.cs_credentials
 module, 50
dfvfs.credentials.encrypted_stream_credentials
 module, 50
dfvfs.credentials.keychain
 module, 50
dfvfs.credentials.luksde_credentials
 module, 51
dfvfs.credentials.manager
 module, 52
dfvfs.encoding
 module, 55
dfvfs.encoding.base16_decoder
 module, 53
dfvfs.encoding.base32_decoder
 module, 53
dfvfs.encoding.base64_decoder
 module, 54
dfvfs.encoding.decoder
 module, 54
dfvfs.encoding.manager
 module, 54
dfvfs.encryption
 module, 58
dfvfs.encryption.aes_decrypter
 module, 55
dfvfs.encryption.blowfish_decrypter
 module, 56
dfvfs.encryption.decrypter
 module, 56
dfvfs.encryption.des3_decrypter
 module, 57
dfvfs.encryption.manager
 module, 57
dfvfs.encryption.rc4_decrypter
 module, 58
dfvfs.file_io
 module, 92
dfvfs.file_io.apfs_file_io
 module, 58
dfvfs.file_io.bde_file_io
 module, 60
dfvfs.file_io.compressed_stream_io
 module, 60
dfvfs.file_io.cpio_file_io
 module, 61
dfvfs.file_io.cs_file_io
 module, 62
dfvfs.file_io.data_range_io
 module, 64
dfvfs.file_io.encoded_stream_io
 module, 65
dfvfs.file_io.encrypted_stream_io
 module, 66
dfvfs.file_io.ewf_file_io
 module, 68
dfvfs.file_io.ext_file_io
 module, 68
dfvfs.file_io.fake_file_io
 module, 69
dfvfs.file_io.fat_file_io

 module, 70
dfvfs.file_io.file_io
 module, 72
dfvfs.file_io.file_object_io
 module, 73
dfvfs.file_io.gpt_file_io
 module, 75
dfvfs.file_io.gzip_file_io
 module, 76
dfvfs.file_io.hfs_file_io
 module, 77
dfvfs.file_io.luksde_file_io
 module, 78
dfvfs.file_io.lvm_file_io
 module, 78
dfvfs.file_io.modi_file_io
 module, 79
dfvfs.file_io.ntfs_file_io
 module, 80
dfvfs.file_io.os_file_io
 module, 81
dfvfs.file_io.phdi_file_io
 module, 82
dfvfs.file_io.qcow_file_io
 module, 82
dfvfs.file_io.raw_file_io
 module, 83
dfvfs.file_io.sqlite_blob_file_io
 module, 83
dfvfs.file_io.tar_file_io
 module, 85
dfvfs.file_io.tsk_file_io
 module, 86
dfvfs.file_io.tsk_partition_file_io
 module, 87
dfvfs.file_io.vhdi_file_io
 module, 87
dfvfs.file_io.vmdk_file_io
 module, 88
dfvfs.file_io.vshadow_file_io
 module, 88
dfvfs.file_io.xfs_file_io
 module, 89
dfvfs.file_io.zip_file_io
 module, 90
dfvfs.helpers
 module, 113
dfvfs.helpers.command_line
 module, 92
dfvfs.helpers.data_slice
 module, 95
dfvfs.helpers.fake_file_system_builder
 module, 96
dfvfs.helpers.file_system_searcher
 module, 97
dfvfs.helpers.source_scanner
 module, 100
dfvfs.helpers.text_file
 module, 107
dfvfs.helpers.volume_scanner
 module, 108
dfvfs.helpers.windows_path_resolver
 module, 112
dfvfs.lib
 module, 127
dfvfs.lib.apfs_helper
 module, 113
dfvfs.lib.bde_helper
 module, 113
dfvfs.lib.cpio
 module, 114
dfvfs.lib.cs_helper
 module, 116
dfvfs.lib.data_format
 module, 117
dfvfs.lib.decorators
 module, 117
dfvfs.lib.definitions
 module, 117
dfvfs.lib.errors
 module, 117
dfvfs.lib.ewf_helper
 module, 118
dfvfs.lib.glob2regex
 module, 119
dfvfs.lib.gpt_helper
 module, 119
dfvfs.lib.gzipfile
 module, 119
dfvfs.lib.luksde_helper
 module, 122
dfvfs.lib.lvm_helper
 module, 123
dfvfs.lib.raw_helper
 module, 123
dfvfs.lib.sqlite_database
 module, 123
dfvfs.lib.tsk_image
 module, 125
dfvfs.lib.tsk_partition
 module, 126
dfvfs.lib.vshadow_helper
 module, 127
dfvfs.mount
 module, 128
dfvfs.mount.manager
 module, 127
dfvfs.path

module, 146
dfvfs.path.apfs_container_path_spec
 module, 128
dfvfs.path.apfs_path_spec
 module, 129
dfvfs.path.bde_path_spec
 module, 129
dfvfs.path.compressed_stream_path_spec
 module, 130
dfvfs.path.cpio_path_spec
 module, 130
dfvfs.path.cs_path_spec
 module, 131
dfvfs.path.data_range_path_spec
 module, 131
dfvfs.path.encoded_stream_path_spec
 module, 132
dfvfs.path.encrypted_stream_path_spec
 module, 132
dfvfs.path.ewf_path_spec
 module, 133
dfvfs.path.ext_path_spec
 module, 133
dfvfs.path.factory
 module, 134
dfvfs.path.fake_path_spec
 module, 135
dfvfs.path.fat_path_spec
 module, 135
dfvfs.path.gpt_path_spec
 module, 136
dfvfs.path.gzip_path_spec
 module, 136
dfvfs.path.hfs_path_spec
 module, 137
dfvfs.path.location_path_spec
 module, 137
dfvfs.path.luksde_path_spec
 module, 138
dfvfs.path.lvm_path_spec
 module, 138
dfvfs.path.modi_path_spec
 module, 139
dfvfs.path.mount_path_spec
 module, 139
dfvfs.path.ntfs_path_spec
 module, 139
dfvfs.path.os_path_spec
 module, 140
dfvfs.path.path_spec
 module, 140
dfvfs.path.phdi_path_spec
 module, 142
dfvfs.path.qcow_path_spec
 module, 142
dfvfs.path.raw_path_spec
 module, 142
dfvfs.path.sqlite_blob_path_spec
 module, 142
dfvfs.path.tar_path_spec
 module, 143
dfvfs.path.tsk_partition_path_spec
 module, 143
dfvfs.path.tsk_path_spec
 module, 144
dfvfs.path.vhdi_path_spec
 module, 145
dfvfs.path.vmdk_path_spec
 module, 145
dfvfs.path.vshadow_path_spec
 module, 145
dfvfs.path.xfs_path_spec
 module, 146
dfvfs.path.zip_path_spec
 module, 146
dfvfs.resolver
 module, 149
dfvfs.resolver.context
 module, 146
dfvfs.resolver.resolver
 module, 148
dfvfs.resolver_helpers
 module, 171
dfvfs.resolver_helpers.apfs_container_resolver_helper
 module, 149
dfvfs.resolver_helpers.apfs_resolver_helper
 module, 150
dfvfs.resolver_helpers.bde_resolver_helper
 module, 151
dfvfs.resolver_helpers.compressed_stream_resolver_helper
 module, 151
dfvfs.resolver_helpers.cpio_resolver_helper
 module, 152
dfvfs.resolver_helpers.cs_resolver_helper
 module, 153
dfvfs.resolver_helpers.data_range_resolver_helper
 module, 153
dfvfs.resolver_helpers.encoded_stream_resolver_helper
 module, 154
dfvfs.resolver_helpers.encrypted_stream_resolver_helper
 module, 155
dfvfs.resolver_helpers.ewf_resolver_helper
 module, 155
dfvfs.resolver_helpers.ext_resolver_helper
 module, 156
dfvfs.resolver_helpers.fake_resolver_helper
 module, 156
dfvfs.resolver_helpers.fat_resolver_helper

module, 157
 dfvfs.resolver_helpers.gpt_resolver_helper
 module, 157
 dfvfs.resolver_helpers.gzip_resolver_helper
 module, 158
 dfvfs.resolver_helpers.hfs_resolver_helper
 module, 159
 dfvfs.resolver_helpers.luksde_resolver_helper
 module, 159
 dfvfs.resolver_helpers.lvm_resolver_helper
 module, 160
 dfvfs.resolver_helpers.manager
 module, 161
 dfvfs.resolver_helpers.modi_resolver_helper
 module, 161
 dfvfs.resolver_helpers.ntfs_resolver_helper
 module, 162
 dfvfs.resolver_helpers.os_resolver_helper
 module, 162
 dfvfs.resolver_helpers.phdi_resolver_helper
 module, 163
 dfvfs.resolver_helpers.qcow_resolver_helper
 module, 164
 dfvfs.resolver_helpers.raw_resolver_helper
 module, 164
 dfvfs.resolver_helpers.resolver_helper
 module, 165
 dfvfs.resolver_helpers.sqlite_blob_resolver_helper
 module, 165
 dfvfs.resolver_helpers.tar_resolver_helper
 module, 166
 dfvfs.resolver_helpers.tsk_partition_resolver_helper
 module, 167
 dfvfs.resolver_helpers.tsk_resolver_helper
 module, 167
 dfvfs.resolver_helpers.vhdi_resolver_helper
 module, 168
 dfvfs.resolver_helpers.vmdk_resolver_helper
 module, 169
 dfvfs.resolver_helpers.vshadow_resolver_helper
 module, 169
 dfvfs.resolver_helpers.xfs_resolver_helper
 module, 170
 dfvfs.resolver_helpers.zip_resolver_helper
 module, 170
 dfvfs.serializer
 module, 172
 dfvfs.serializer.json_serializer
 module, 171
 dfvfs.serializer.serializer
 module, 172
 dfvfs.vfs
 module, 253
 dfvfs.vfs.apfs_attribute
 module, 172
 dfvfs.vfs.apfs_container_directory
 module, 174
 dfvfs.vfs.apfs_container_file_entry
 module, 174
 dfvfs.vfs.apfs_container_file_system
 module, 175
 dfvfs.vfs.apfs_directory
 module, 176
 dfvfs.vfs.apfs_file_entry
 module, 176
 dfvfs.vfs.apfs_file_system
 module, 178
 dfvfs.vfs.attribute
 module, 179
 dfvfs.vfs.bde_file_entry
 module, 181
 dfvfs.vfs.bde_file_system
 module, 181
 dfvfs.vfs.compressed_stream_file_entry
 module, 182
 dfvfs.vfs.compressed_stream_file_system
 module, 182
 dfvfs.vfs.cpio_directory
 module, 183
 dfvfs.vfs.cpio_file_entry
 module, 183
 dfvfs.vfs.cpio_file_system
 module, 184
 dfvfs.vfs.cs_directory
 module, 185
 dfvfs.vfs.cs_file_entry
 module, 185
 dfvfs.vfs.cs_file_system
 module, 186
 dfvfs.vfs.data_range_file_entry
 module, 187
 dfvfs.vfs.data_range_file_system
 module, 188
 dfvfs.vfs.data_stream
 module, 188
 dfvfs.vfs.directory
 module, 189
 dfvfs.vfs.encoded_stream_file_entry
 module, 189
 dfvfs.vfs.encoded_stream_file_system
 module, 189
 dfvfs.vfs.encrypted_stream_file_entry
 module, 190
 dfvfs.vfs.encrypted_stream_file_system
 module, 190
 dfvfs.vfs.ext_attribute
 module, 191
 dfvfs.vfs.ext_directory

module, 192
dfvfs.vfs.ext_file_entry
 module, 193
dfvfs.vfs.ext_file_system
 module, 194
dfvfs.vfs.extent
 module, 195
dfvfs.vfs.fake_directory
 module, 196
dfvfs.vfs.fake_file_entry
 module, 196
dfvfs.vfs.fake_file_system
 module, 197
dfvfs.vfs.fat_directory
 module, 199
dfvfs.vfs.fat_file_entry
 module, 199
dfvfs.vfs.fat_file_system
 module, 200
dfvfs.vfs.file_entry
 module, 202
dfvfs.vfs.file_system
 module, 207
dfvfs.vfs.gpt_directory
 module, 210
dfvfs.vfs.gpt_file_entry
 module, 210
dfvfs.vfs.gpt_file_system
 module, 211
dfvfs.vfs.gzip_file_entry
 module, 212
dfvfs.vfs.gzip_file_system
 module, 212
dfvfs.vfs.hfs_attribute
 module, 213
dfvfs.vfs.hfs_data_stream
 module, 214
dfvfs.vfs.hfs_directory
 module, 215
dfvfs.vfs.hfs_file_entry
 module, 215
dfvfs.vfs.hfs_file_system
 module, 217
dfvfs.vfs.luksde_file_entry
 module, 218
dfvfs.vfs.luksde_file_system
 module, 218
dfvfs.vfs.lvm_directory
 module, 219
dfvfs.vfs.lvm_file_entry
 module, 219
dfvfs.vfs.lvm_file_system
 module, 220
dfvfs.vfs.ntfs_attribute
 module, 221
dfvfs.vfs.ntfs_data_stream
 module, 223
dfvfs.vfs.ntfs_directory
 module, 224
dfvfs.vfs.ntfs_file_entry
 module, 224
dfvfs.vfs.ntfs_file_system
 module, 226
dfvfs.vfs.os_attribute
 module, 227
dfvfs.vfs.os_directory
 module, 228
dfvfs.vfs.os_file_entry
 module, 229
dfvfs.vfs.os_file_system
 module, 230
dfvfs.vfs.root_only_file_entry
 module, 231
dfvfs.vfs.root_only_file_system
 module, 231
dfvfs.vfs.sqlite_blob_directory
 module, 232
dfvfs.vfs.sqlite_blob_file_entry
 module, 232
dfvfs.vfs.sqlite_blob_file_system
 module, 233
dfvfs.vfs.tar_directory
 module, 233
dfvfs.vfs.tar_file_entry
 module, 234
dfvfs.vfs.tar_file_system
 module, 235
dfvfs.vfs.tsk_attribute
 module, 236
dfvfs.vfs.tsk_data_stream
 module, 237
dfvfs.vfs.tsk_directory
 module, 238
dfvfs.vfs.tsk_file_entry
 module, 238
dfvfs.vfs.tsk_file_system
 module, 241
dfvfs.vfs.tsk_partition_directory
 module, 243
dfvfs.vfs.tsk_partition_file_entry
 module, 243
dfvfs.vfs.tsk_partition_file_system
 module, 244
dfvfs.vfs.vshadow_directory
 module, 245
dfvfs.vfs.vshadow_file_entry
 module, 245
dfvfs.vfs.vshadow_file_system

module, 246
 dfvfs.vfs.xfs_attribute
 module, 247
 dfvfs.vfs.xfs_directory
 module, 248
 dfvfs.vfs.xfs_file_entry
 module, 248
 dfvfs.vfs.xfs_file_system
 module, 250
 dfvfs.vfs.zip_directory
 module, 251
 dfvfs.vfs.zip_file_entry
 module, 251
 dfvfs.vfs.zip_file_system
 module, 252
 dfvfs.volume
 module, 259
 dfvfs.volume.apfs_volume_system
 module, 253
 dfvfs.volume.cs_volume_system
 module, 254
 dfvfs.volume.factory
 module, 254
 dfvfs.volume.gpt_volume_system
 module, 255
 dfvfs.volume.lvm_volume_system
 module, 255
 dfvfs.volume.tsk_volume_system
 module, 255
 dfvfs.volume.volume_system
 module, 256
 dfvfs.volume.vshadow_volume_system
 module, 258
 Directory (class in *dfvfs.vfs.directory*), 189
 DirnamePath() (*dfvfs.vfs.file_system.FileSystem*
 method), 207
 droid_file_identifier
 (*dfvfs.vfs.ntfs_attribute.ObjectIdentifierNTFSAttribute*
 property), 222

E

Empty() (*dfvfs.credentials.keychain.KeyChain* method),
 50
 Empty() (*dfvfs.resolver.context.Context* method), 147
 EncodedStream (class in
 dfvfs.file_io.encoded_stream_io), 65
 EncodedStreamFileEntry (class in
 dfvfs.vfs.encoded_stream_file_entry), 189
 EncodedStreamFileSystem (class in
 dfvfs.vfs.encoded_stream_file_system), 189
 EncodedStreamPathSpec (class in
 dfvfs.path.encoded_stream_path_spec), 132
 EncodedStreamResolverHelper (class in
 dfvfs.resolver_helpers.encoded_stream_resolver_helper),
 154
 encoding (*dfvfs.lib.cpio.CPIOArchiveFile* property), 115
 encoding (*dfvfs.vfs.tar_file_system.TARFileSystem* at-
 tribute), 235
 encoding (*dfvfs.vfs.zip_file_system.ZipFileSystem*
 attribute), 252
 ENCODING_METHOD (*dfvfs.encoding.base16_decoder.Base16Decoder*
 attribute), 53
 ENCODING_METHOD (*dfvfs.encoding.base32_decoder.Base32Decoder*
 attribute), 53
 ENCODING_METHOD (*dfvfs.encoding.base64_decoder.Base64Decoder*
 attribute), 54
 encoding_method (*dfvfs.path.encoded_stream_path_spec.EncodedStream*
 attribute), 132
 EncodingManager (class in *dfvfs.encoding.manager*), 54
 encrypted_root_plist
 (*dfvfs.path.cs_path_spec.CSPathSpec* at-
 tribute), 131
 EncryptedStream (class in
 dfvfs.file_io.encrypted_stream_io), 66
 EncryptedStreamCredentials (class in
 dfvfs.credentials.encrypted_stream_credentials),
 50
 EncryptedStreamFileEntry (class in
 dfvfs.vfs.encrypted_stream_file_entry), 190
 EncryptedStreamFileSystem (class in
 dfvfs.vfs.encrypted_stream_file_system), 190
 EncryptedStreamPathSpec (class in
 dfvfs.path.encrypted_stream_path_spec),
 132
 EncryptedStreamResolverHelper (class in
 dfvfs.resolver_helpers.encrypted_stream_resolver_helper),
 155
 ENCRYPTION_METHOD (*dfvfs.encryption.aes_decrypter.AESDecrypter*
 attribute), 55
 ENCRYPTION_METHOD (*dfvfs.encryption.blowfish_decrypter.BlowfishDecrypter*
 attribute), 56
 ENCRYPTION_METHOD (*dfvfs.encryption.des3_decrypter.DES3Decrypter*
 attribute), 57
 ENCRYPTION_METHOD (*dfvfs.encryption.rc4_decrypter.RC4Decrypter*
 attribute), 58
 encryption_method (*dfvfs.path.encrypted_stream_path_spec.EncryptedStream*
 attribute), 133
 EncryptionManager (class in
 dfvfs.encryption.manager), 57
 entries (*dfvfs.vfs.directory.Directory* property), 189
 entry_index (*dfvfs.path.gpt_path_spec.GPTPathSpec*
 attribute), 136
 entry_modification_time
 (*dfvfs.vfs.ntfs_attribute.FileNameNTFSAttribute*
 property), 221
 entry_modification_time
 (*dfvfs.vfs.ntfs_attribute.StandardInformationNTFSAttribute*
 property), 223

- entry_type (*dfvfs.vfs.file_entry.FileEntry* attribute), 202
- Error, 117
- EWFAalyzerHelper (class in *dfvfs.analyzer.ewf_analyzer_helper*), 34
- EWFFile (class in *dfvfs.file_io.ewf_file_io*), 68
- EWFGlobPathSpec() (in module *dfvfs.lib.ewf_helper*), 118
- EWFPPathSpec (class in *dfvfs.path.ewf_path_spec*), 133
- EWFResolverHelper (class in *dfvfs.resolver_helpers.ewf_resolver_helper*), 155
- EXTAnalyzerHelper (class in *dfvfs.analyzer.ext_analyzer_helper*), 34
- EXTDirectory (class in *dfvfs.vfs.ext_directory*), 192
- Extent (class in *dfvfs.vfs.extent*), 195
- extent_type (*dfvfs.vfs.extent.Extent* attribute), 195
- EXTENT_TYPE_DATA (*dfvfs.volume.volume_system.VolumeExtent* attribute), 257
- EXTENT_TYPE_SPARSE (*dfvfs.volume.volume_system.VolumeExtent* attribute), 257
- extents (*dfvfs.volume.volume_system.Volume* property), 256
- EXTExtendedAttribute (class in *dfvfs.vfs.ext_attribute*), 191
- EXTFile (class in *dfvfs.file_io.ext_file_io*), 68
- EXTFileEntry (class in *dfvfs.vfs.ext_file_entry*), 193
- EXTFileSystem (class in *dfvfs.vfs.ext_file_system*), 194
- EXTPathSpec (class in *dfvfs.path.ext_path_spec*), 133
- ExtractCredentialsFromPathSpec() (*dfvfs.credentials.keychain.KeyChain* method), 50
- EXTResolverHelper (class in *dfvfs.resolver_helpers.ext_resolver_helper*), 156
- F**
- Factory (class in *dfvfs.path.factory*), 134
- Factory (class in *dfvfs.volume.factory*), 254
- FakeDirectory (class in *dfvfs.vfs.fake_directory*), 196
- FakeFile (class in *dfvfs.file_io.fake_file_io*), 69
- FakeFileEntry (class in *dfvfs.vfs.fake_file_entry*), 196
- FakeFileSystem (class in *dfvfs.vfs.fake_file_system*), 197
- FakeFileSystemBuilder (class in *dfvfs.helpers.fake_file_system_builder*), 96
- FakePathSpec (class in *dfvfs.path.fake_path_spec*), 135
- FakeResolverHelper (class in *dfvfs.resolver_helpers.fake_resolver_helper*), 156
- FATANalyzerHelper (class in *dfvfs.analyzer.fat_analyzer_helper*), 35
- FATDirectory (class in *dfvfs.vfs.fat_directory*), 199
- FATFile (class in *dfvfs.file_io.fat_file_io*), 70
- FATFileEntry (class in *dfvfs.vfs.fat_file_entry*), 199
- FATFileSystem (class in *dfvfs.vfs.fat_file_system*), 200
- FATPathSpec (class in *dfvfs.path.fat_path_spec*), 135
- FATResolverHelper (class in *dfvfs.resolver_helpers.fat_resolver_helper*), 157
- file_attribute_flags (*dfvfs.vfs.ntfs_attribute.FileNameNTFSAttribute* property), 221
- file_attribute_flags (*dfvfs.vfs.ntfs_attribute.StandardInformationNTFSAttribute* property), 223
- file_format (*dfvfs.lib.cpio.CPIOArchiveFile* attribute), 114
- file_object (*dfvfs.lib.cpio.CPIOArchiveFile* attribute), 115
- file_object (*dfvfs.lib.zipfile.GzipMember* attribute), 122
- file_system (*dfvfs.helpers.fake_file_system_builder.FakeFileSystemBuilder* attribute), 202
- FileEntry (class in *dfvfs.vfs.file_entry*), 202
- FileEntryExistsByPath() (*dfvfs.lib.cpio.CPIOArchiveFile* method), 114
- FileEntryExistsByPath() (*dfvfs.vfs.fake_file_system.FakeFileSystem* method), 197
- FileEntryExistsByPathSpec() (*dfvfs.vfs.apfs_container_file_system.APFSContainerFileSystem* method), 175
- FileEntryExistsByPathSpec() (*dfvfs.vfs.apfs_file_system.APFSSystem* method), 178
- FileEntryExistsByPathSpec() (*dfvfs.vfs.cpio_file_system.CPIOFileSystem* method), 184
- FileEntryExistsByPathSpec() (*dfvfs.vfs.cs_file_system.CSFileSystem* method), 186
- FileEntryExistsByPathSpec() (*dfvfs.vfs.ext_file_system.EXTFileSystem* method), 194
- FileEntryExistsByPathSpec() (*dfvfs.vfs.fake_file_system.FakeFileSystem* method), 198
- FileEntryExistsByPathSpec() (*dfvfs.vfs.fat_file_system.FATFileSystem* method), 200
- FileEntryExistsByPathSpec() (*dfvfs.vfs.file_system.FileSystem* method), 208
- FileEntryExistsByPathSpec() (*dfvfs.vfs.gpt_file_system.GPTFileSystem* method), 211
- FileEntryExistsByPathSpec()

(*dfvfs.vfs.hfs_file_system.HFSFileSystem* method), 217

FileEntryExistsByPathSpec() (*dfvfs.vfs.lvm_file_system.LVMFileSystem* method), 220

FileEntryExistsByPathSpec() (*dfvfs.vfs.ntfs_file_system.NTFSFileSystem* method), 226

FileEntryExistsByPathSpec() (*dfvfs.vfs.os_file_system.OSFileSystem* method), 230

FileEntryExistsByPathSpec() (*dfvfs.vfs.root_only_file_system.RootOnlyFileSystem* method), 231

FileEntryExistsByPathSpec() (*dfvfs.vfs.sqlite_blob_file_system.SQLiteBlobFileSystem* method), 233

FileEntryExistsByPathSpec() (*dfvfs.vfs.tar_file_system.TARFileSystem* method), 235

FileEntryExistsByPathSpec() (*dfvfs.vfs.tsk_file_system.TSKFileSystem* method), 241

FileEntryExistsByPathSpec() (*dfvfs.vfs.tsk_partition_file_system.TSKPartitionFileSystem* method), 244

FileEntryExistsByPathSpec() (*dfvfs.vfs.vshadow_file_system.VShadowFileSystem* method), 246

FileEntryExistsByPathSpec() (*dfvfs.vfs.xfs_file_system.XFSFileSystem* method), 250

FileEntryExistsByPathSpec() (*dfvfs.vfs.zip_file_system.ZipFileSystem* method), 252

FileFormatError, 117

FileIO (class in *dfvfs.file_io.file_io*), 72

FileNameNTFSAttribute (class in *dfvfs.vfs.ntfs_attribute*), 221

FileObjectInputReader (class in *dfvfs.helpers.command_line*), 94

FileObjectIO (class in *dfvfs.file_io.file_object_io*), 73

FileObjectOutputWriter (class in *dfvfs.helpers.command_line*), 95

FileSystem (class in *dfvfs.vfs.file_system*), 207

FileSystemSearcher (class in *dfvfs.helpers.file_system_searcher*), 97

Find() (*dfvfs.helpers.file_system_searcher.FileSystemSearcher* method), 97

FindSpec (class in *dfvfs.helpers.file_system_searcher*), 98

Flush() (*dfvfs.helpers.command_line.CLIOutputWriter* method), 92

Flush() (*dfvfs.helpers.command_line.StdoutOutputWriter* method), 95

FlushCache() (*dfvfs.lib.gzipfile.GzipMember* method), 122

format_categories (*dfvfs.analyzer.analyzer_helper.AnalyzerHelper* property), 31

FORMAT_CATEGORIES (*dfvfs.analyzer.apfs_analyzer_helper.APFSAnalyzerHelper* attribute), 32

FORMAT_CATEGORIES (*dfvfs.analyzer.apfs_container_analyzer_helper.APFSContainerAnalyzerHelper* attribute), 32

FORMAT_CATEGORIES (*dfvfs.analyzer.bde_analyzer_helper.BDEAnalyzerHelper* attribute), 32

FORMAT_CATEGORIES (*dfvfs.analyzer.bzip2_analyzer_helper.BZIP2AnalyzerHelper* attribute), 33

FORMAT_CATEGORIES (*dfvfs.analyzer.cpio_analyzer_helper.CPIOAnalyzerHelper* attribute), 33

FORMAT_CATEGORIES (*dfvfs.analyzer.cs_analyzer_helper.CSAnalyzerHelper* attribute), 34

FORMAT_CATEGORIES (*dfvfs.analyzer.ewf_analyzer_helper.EWFAnalyzerHelper* attribute), 34

FORMAT_CATEGORIES (*dfvfs.analyzer.ext_analyzer_helper.EXTAnalyzerHelper* attribute), 34

FORMAT_CATEGORIES (*dfvfs.analyzer.fat_analyzer_helper.FATAnalyzerHelper* attribute), 35

FORMAT_CATEGORIES (*dfvfs.analyzer.gpt_analyzer_helper.GPTAnalyzerHelper* attribute), 36

FORMAT_CATEGORIES (*dfvfs.analyzer.gzip_analyzer_helper.GzipAnalyzerHelper* attribute), 36

FORMAT_CATEGORIES (*dfvfs.analyzer.hfs_analyzer_helper.HFSAnalyzerHelper* attribute), 37

FORMAT_CATEGORIES (*dfvfs.analyzer.luksde_analyzer_helper.LUKSDEAnalyzerHelper* attribute), 37

FORMAT_CATEGORIES (*dfvfs.analyzer.lvm_analyzer_helper.LVMAnalyzerHelper* attribute), 38

FORMAT_CATEGORIES (*dfvfs.analyzer.modi_analyzer_helper.MODIAnalyzerHelper* attribute), 38

FORMAT_CATEGORIES (*dfvfs.analyzer.ntfs_analyzer_helper.NTFSAnalyzerHelper* attribute), 38

FORMAT_CATEGORIES (*dfvfs.analyzer.phdi_analyzer_helper.PHDIAnalyzerHelper* attribute), 39

FORMAT_CATEGORIES (*dfvfs.analyzer.qcow_analyzer_helper.QCOWAnalyzerHelper* attribute), 39

FORMAT_CATEGORIES (*dfvfs.analyzer.tar_analyzer_helper.TARAnalyzerHelper* attribute), 42

FORMAT_CATEGORIES (*dfvfs.analyzer.tsk_analyzer_helper.TSKAnalyzerHelper* attribute), 42

FORMAT_CATEGORIES (*dfvfs.analyzer.tsk_partition_analyzer_helper.TSKPartitionAnalyzerHelper* attribute), 43

FORMAT_CATEGORIES (*dfvfs.analyzer.vhdi_analyzer_helper.VHDIAnalyzerHelper* attribute), 43

FORMAT_CATEGORIES (*dfvfs.analyzer.vmdk_analyzer_helper.VMDKAnalyzerHelper* attribute), 43

FORMAT_CATEGORIES (*dfvfs.analyzer.vshadow_analyzer_helper.VShadowAnalyzerHelper* attribute), 44

FORMAT_CATEGORIES (*dfvfs.analyzer.xfs_analyzer_helper.XFSAnalyzerHelper* attribute), 44

- attribute), 44
- FORMAT_CATEGORIES (dfvfs.analyzer.xz_analyzer_helper.XZAnalyzerHelper attribute), 45
- FORMAT_CATEGORIES (dfvfs.analyzer.zip_analyzer_helper.ZipAnalyzerHelper attribute), 45
- FormatSpecification (class in dfvfs.analyzer.specification), 40
- FormatSpecificationStore (class in dfvfs.analyzer.specification), 40
- fraction_of_second (dfvfs.vfs.task_file_entry.TSKTime attribute), 240
- ## G
- get_offset() (dfvfs.file_io.apfs_file_io.APFSFile method), 58
- get_offset() (dfvfs.file_io.compressed_stream_io.CompressedStream method), 60
- get_offset() (dfvfs.file_io.cpio_file_io.CPIOFile method), 61
- get_offset() (dfvfs.file_io.cs_file_io.CSFile method), 62
- get_offset() (dfvfs.file_io.data_range_io.DataRange method), 64
- get_offset() (dfvfs.file_io.encoded_stream_io.EncodedStream method), 65
- get_offset() (dfvfs.file_io.encrypted_stream_io.EncryptedStream method), 67
- get_offset() (dfvfs.file_io.ext_file_io.EXTFile method), 68
- get_offset() (dfvfs.file_io.fake_file_io.FakeFile method), 69
- get_offset() (dfvfs.file_io.fat_file_io.FATFile method), 70
- get_offset() (dfvfs.file_io.file_io.FileIO method), 72
- get_offset() (dfvfs.file_io.file_object_io.FileObjectIO method), 73
- get_offset() (dfvfs.file_io.gpt_file_io.GPTFile method), 75
- get_offset() (dfvfs.file_io.hfs_file_io.HFSFile method), 77
- get_offset() (dfvfs.file_io.lvm_file_io.LVMFile method), 78
- get_offset() (dfvfs.file_io.ntfs_file_io.NTFSFile method), 80
- get_offset() (dfvfs.file_io.os_file_io.OSFile method), 81
- get_offset() (dfvfs.file_io.sqlite_blob_file_io.SQLiteBlobFile method), 83
- get_offset() (dfvfs.file_io.tar_file_io.TARFile method), 85
- get_offset() (dfvfs.file_io.task_file_io.TSKFile method), 86
- get_offset() (dfvfs.file_io.vshadow_file_io.VShadowFile method), 88
- get_offset() (dfvfs.file_io.xfs_file_io.XFSFile method), 89
- get_offset() (dfvfs.file_io.zip_file_io.ZipFile method), 90
- get_offset() (dfvfs.helpers.text_file.TextFile method), 107
- get_offset() (dfvfs.lib.gzipfile.GzipCompressedStream method), 120
- get_offset() (dfvfs.vfs.apfs_attribute.APFSExtendedAttribute method), 173
- get_offset() (dfvfs.vfs.ext_attribute.EXTExtendedAttribute method), 191
- get_offset() (dfvfs.vfs.hfs_attribute.HFSExtendedAttribute method), 213
- get_offset() (dfvfs.vfs.os_attribute.OSExtendedAttribute method), 227
- get_offset() (dfvfs.vfs.task_attribute.TSKExtendedAttribute method), 236
- get_offset() (dfvfs.vfs.xfs_attribute.XFSExtendedAttribute method), 247
- get_size() (dfvfs.file_io.apfs_file_io.APFSFile method), 59
- get_size() (dfvfs.file_io.compressed_stream_io.CompressedStream method), 60
- get_size() (dfvfs.file_io.cpio_file_io.CPIOFile method), 61
- get_size() (dfvfs.file_io.cs_file_io.CSFile method), 62
- get_size() (dfvfs.file_io.data_range_io.DataRange method), 64
- get_size() (dfvfs.file_io.encoded_stream_io.EncodedStream method), 65
- get_size() (dfvfs.file_io.encrypted_stream_io.EncryptedStream method), 67
- get_size() (dfvfs.file_io.ewf_file_io.EWFFile method), 68
- get_size() (dfvfs.file_io.ext_file_io.EXTFile method), 68
- get_size() (dfvfs.file_io.fake_file_io.FakeFile method), 69
- get_size() (dfvfs.file_io.fat_file_io.FATFile method), 71
- get_size() (dfvfs.file_io.file_io.FileIO method), 72
- get_size() (dfvfs.file_io.file_object_io.FileObjectIO method), 74
- get_size() (dfvfs.file_io.gpt_file_io.GPTFile method), 75
- get_size() (dfvfs.file_io.gzip_file_io.GzipFile method), 76
- get_size() (dfvfs.file_io.hfs_file_io.HFSFile method), 77
- get_size() (dfvfs.file_io.lvm_file_io.LVMFile method), 78
- get_size() (dfvfs.file_io.modi_file_io.MODIFile method), 79

get_size() (dfvfs.file_io.ntfs_file_io.NTFSFile method), 80
 get_size() (dfvfs.file_io.os_file_io.OSFile method), 81
 get_size() (dfvfs.file_io.phdi_file_io.PHDIFile method), 82
 get_size() (dfvfs.file_io.qcow_file_io.QCOWFile method), 82
 get_size() (dfvfs.file_io.raw_file_io.RawFile method), 83
 get_size() (dfvfs.file_io.sqlite_blob_file_io.SQLiteBlobFile method), 84
 get_size() (dfvfs.file_io.tar_file_io.TARFile method), 85
 get_size() (dfvfs.file_io.tsk_file_io.TSKFile method), 86
 get_size() (dfvfs.file_io.vhdi_file_io.VHDIFile method), 87
 get_size() (dfvfs.file_io.vmdk_file_io.VMDKFile method), 88
 get_size() (dfvfs.file_io.vshadow_file_io.VShadowFile method), 88
 get_size() (dfvfs.file_io.xfs_file_io.XFSFile method), 89
 get_size() (dfvfs.file_io.zip_file_io.ZipFile method), 91
 get_size() (dfvfs.lib.gzipfile.GzipCompressedStream method), 120
 get_size() (dfvfs.lib.tsk_image.TSKFileSystemImage method), 125
 get_size() (dfvfs.vfs.apfs_attribute.APFSExtendedAttribute method), 173
 get_size() (dfvfs.vfs.ext_attribute.EXTExtendedAttribute method), 191
 get_size() (dfvfs.vfs.hfs_attribute.HFSExtendedAttribute method), 213
 get_size() (dfvfs.vfs.os_attribute.OSExtendedAttribute method), 227
 get_size() (dfvfs.vfs.tsk_attribute.TSKExtendedAttribute method), 236
 get_size() (dfvfs.vfs.xfs_attribute.XFSExtendedAttribute method), 247
 GetAPFSContainer() (dfvfs.vfs.apfs_container_file_system.APFSContainerFileSystem method), 175
 GetAPFSFileEntry() (dfvfs.vfs.apfs_file_entry.APFSFileEntry method), 176
 GetAPFSFileEntryByPathSpec() (dfvfs.vfs.apfs_file_system.APFSFileSystem method), 178
 GetAPFSVolume() (dfvfs.vfs.apfs_container_file_entry.APFSContainerFileEntry method), 174
 GetAPFSVolumeByPathSpec() (dfvfs.vfs.apfs_container_file_system.APFSContainerFileSystem method), 175
 GetAPFSVolumeIdentifiers() (dfvfs.helpers.command_line.CLIVolumeScannerMethod method), 93
 GetAPFSVolumeIdentifiers() (dfvfs.helpers.volume_scanner.VolumeScannerMediator method), 108
 GetArchiveTypeIndicators() (dfvfs.analyzer.analyzer.Analyzer class method), 29
 GetAttribute() (dfvfs.volume.volume_system.Volume method), 256
 GetBasePathSpecs() (dfvfs.helpers.volume_scanner.VolumeScanner method), 108
 GetBDEVVolume() (dfvfs.vfs.bde_file_system.BDEFileSystem method), 181
 GetCompressedStreamTypeIndicators() (dfvfs.analyzer.analyzer.Analyzer class method), 29
 GetCPIOArchiveFile() (dfvfs.vfs.cpio_file_system.CPIOFileSystem method), 184
 GetCPIOArchiveFileEntry() (dfvfs.vfs.cpio_file_entry.CPIOFileEntry method), 183
 GetCPIOArchiveFileEntryByPathSpec() (dfvfs.vfs.cpio_file_system.CPIOFileSystem method), 184
 GetCredential() (dfvfs.credentials.keychain.KeyChain method), 50
 GetCredentials() (dfvfs.credentials.keychain.KeyChain method), 51
 GetCredentials() (dfvfs.credentials.manager.CredentialsManager class method), 52
 GetDataByPath() (dfvfs.vfs.fake_file_system.FakeFileSystem method), 198
 GetDataStream() (dfvfs.vfs.file_entry.FileEntry method), 202
 GetDataStreamByPathSpec() (dfvfs.vfs.file_system.FileSystem method), 208
 GetDate() (dfvfs.vfs.tsk_file_entry.TSKTime method), 240
 GetEncodingManager() (dfvfs.encoding.manager.EncodingManager class method), 55
 GetDecompressor() (dfvfs.compression.manager.CompressionManager class method), 47
 GetDecrypter() (dfvfs.encryption.manager.EncryptionManager class method), 57
 GetExtents() (dfvfs.vfs.apfs_attribute.APFSExtendedAttribute method), 172
 GetExtents() (dfvfs.vfs.apfs_file_entry.APFSFileEntry method), 177
 GetExtents() (dfvfs.vfs.attribute.Attribute method), 179
 GetExtents() (dfvfs.vfs.data_stream.DataStream method), 188
 GetExtents() (dfvfs.vfs.ext_attribute.EXTExtendedAttribute

method), 191
 GetExtents() (*dfvfs.vfs.ext_file_entry.EXTFileEntry method*), 193
 GetExtents() (*dfvfs.vfs.fat_file_entry.FATFileEntry method*), 199
 GetExtents() (*dfvfs.vfs.file_entry.FileEntry method*), 202
 GetExtents() (*dfvfs.vfs.hfs_attribute.HFSExtendedAttribute method*), 213
 GetExtents() (*dfvfs.vfs.hfs_data_stream.HFSDataStream method*), 214
 GetExtents() (*dfvfs.vfs.hfs_file_entry.HFSFileEntry method*), 215
 GetExtents() (*dfvfs.vfs.ntfs_data_stream.NTFSDataStream method*), 223
 GetExtents() (*dfvfs.vfs.ntfs_file_entry.NTFSTFileEntry method*), 224
 GetExtents() (*dfvfs.vfs.tsk_data_stream.TSKDataStream method*), 237
 GetExtents() (*dfvfs.vfs.tsk_file_entry.TSKFileEntry method*), 238
 GetExtents() (*dfvfs.vfs.xfs_attribute.XFSExtendedAttribute method*), 247
 GetExtents() (*dfvfs.vfs.xfs_file_entry.XFSFileEntry method*), 248
 GetEXTFileEntry() (*dfvfs.vfs.ext_file_entry.EXTFileEntry method*), 193
 GetEXTFileEntryByPathSpec() (*dfvfs.vfs.ext_file_system.EXTFileSystem method*), 194
 GetFATFileEntry() (*dfvfs.vfs.fat_file_entry.FATFileEntry method*), 199
 GetFATFileEntryByPathSpec() (*dfvfs.vfs.fat_file_system.FATFileSystem method*), 201
 GetFATVolume() (*dfvfs.vfs.fat_file_system.FATFileSystem method*), 201
 GetFileEntries() (*dfvfs.lib.cpio.CPIOArchiveFile method*), 114
 GetFileEntryByPath() (*dfvfs.lib.cpio.CPIOArchiveFile method*), 114
 GetFileEntryByPath() (*dfvfs.vfs.fake_file_system.FakeFileSystem method*), 198
 GetFileEntryByPathSpec() (*dfvfs.helpers.file_system_searcher.FileSystemSearcher method*), 97
 GetFileEntryByPathSpec() (*dfvfs.vfs.apfs_container_file_system.APFSContainerFileSystem method*), 176
 GetFileEntryByPathSpec() (*dfvfs.vfs.apfs_file_system.APFSFileSystem method*), 178
 GetFileEntryByPathSpec() (*dfvfs.vfs.bde_file_system.BDEFileSystem method*), 181
 GetFileEntryByPathSpec() (*dfvfs.vfs.compressed_stream_file_system.CompressedStreamFileSystem method*), 182
 GetFileEntryByPathSpec() (*dfvfs.vfs.cpio_file_system.CPIOFileSystem method*), 185
 GetFileEntryByPathSpec() (*dfvfs.vfs.cs_file_system.CSFileSystem method*), 187
 GetFileEntryByPathSpec() (*dfvfs.vfs.data_range_file_system.DataRangeFileSystem method*), 188
 GetFileEntryByPathSpec() (*dfvfs.vfs.encoded_stream_file_system.EncodedStreamFileSystem method*), 189
 GetFileEntryByPathSpec() (*dfvfs.vfs.encrypted_stream_file_system.EncryptedStreamFileSystem method*), 190
 GetFileEntryByPathSpec() (*dfvfs.vfs.ext_file_system.EXTFileSystem method*), 195
 GetFileEntryByPathSpec() (*dfvfs.vfs.fake_file_system.FakeFileSystem method*), 198
 GetFileEntryByPathSpec() (*dfvfs.vfs.fat_file_system.FATFileSystem method*), 201
 GetFileEntryByPathSpec() (*dfvfs.vfs.file_system.FileSystem method*), 208
 GetFileEntryByPathSpec() (*dfvfs.vfs.gpt_file_system.GPTFileSystem method*), 211
 GetFileEntryByPathSpec() (*dfvfs.vfs.zip_file_system.ZipFileSystem method*), 212
 GetFileEntryByPathSpec() (*dfvfs.vfs.hfs_file_system.HFSFileSystem method*), 217
 GetFileEntryByPathSpec() (*dfvfs.vfs.luksde_file_system.LUKSDEFileSystem method*), 218
 GetFileEntryByPathSpec() (*dfvfs.vfs.lvm_file_system.LVMFileSystem method*), 220
 GetFileEntryByPathSpec() (*dfvfs.vfs.ntfs_file_system.NTFSTFileSystem method*), 226
 GetFileEntryByPathSpec() (*dfvfs.vfs.os_file_system.OSFileSystem method*), 230

GetFileEntryByPathSpec() (dfvfs.vfs.root_only_file_system.RootOnlyFileSystem method), 231	GetFormatSpecification() (dfvfs.analyzer.apfs_container_analyzer_helper.APFSContainerA method), 32
GetFileEntryByPathSpec() (dfvfs.vfs.sqlite_blob_file_system.SQLiteBlobFileSystem method), 233	GetFormatSpecification() (dfvfs.analyzer.bde_analyzer_helper.BDEAnalyzerHelper method), 32
GetFileEntryByPathSpec() (dfvfs.vfs.tar_file_system.TARFileSystem method), 235	GetFormatSpecification() (dfvfs.analyzer.bzip2_analyzer_helper.BZIP2AnalyzerHelper method), 33
GetFileEntryByPathSpec() (dfvfs.vfs.tsk_file_system.TSKFileSystem method), 241	GetFormatSpecification() (dfvfs.analyzer.cpio_analyzer_helper.CPIOAnalyzerHelper method), 33
GetFileEntryByPathSpec() (dfvfs.vfs.tsk_partition_file_system.TSKPartitionFileSystem method), 244	GetFormatSpecification() (dfvfs.analyzer.cs_analyzer_helper.CSAnalyzerHelper method), 34
GetFileEntryByPathSpec() (dfvfs.vfs.vshadow_file_system.VShadowFileSystem method), 246	GetFormatSpecification() (dfvfs.analyzer.ewf_analyzer_helper.EWFAnalyzerHelper method), 34
GetFileEntryByPathSpec() (dfvfs.vfs.xfs_file_system.XFSFileSystem method), 250	GetFormatSpecification() (dfvfs.analyzer.ext_analyzer_helper.EXTAnalyzerHelper method), 34
GetFileEntryByPathSpec() (dfvfs.vfs.zip_file_system.ZipFileSystem method), 252	GetFormatSpecification() (dfvfs.analyzer.fat_analyzer_helper.FATAnalyzerHelper method), 35
GetFileObject() (dfvfs.resolver.context.Context method), 147	GetFormatSpecification() (dfvfs.analyzer.gpt_analyzer_helper.GPTAnalyzerHelper method), 36
GetFileObject() (dfvfs.vfs.fake_file_entry.FakeFileEntry method), 196	GetFormatSpecification() (dfvfs.analyzer.gzip_analyzer_helper.GzipAnalyzerHelper method), 36
GetFileObject() (dfvfs.vfs.fat_file_entry.FATFileEntry method), 199	GetFormatSpecification() (dfvfs.analyzer.hfs_analyzer_helper.HFSAnalyzerHelper method), 37
GetFileObject() (dfvfs.vfs.file_entry.FileEntry method), 202	GetFormatSpecification() (dfvfs.analyzer.luksde_analyzer_helper.LUKSDEAnalyzerHelper method), 37
GetFileObject() (dfvfs.vfs.hfs_file_entry.HFSFileEntry method), 215	GetFormatSpecification() (dfvfs.analyzer.lvm_analyzer_helper.LVMAnalyzerHelper method), 38
GetFileObject() (dfvfs.vfs.ntfs_file_entry.NTFSEntry method), 224	GetFormatSpecification() (dfvfs.analyzer.modi_analyzer_helper.MODIAnalyzerHelper method), 38
GetFileObject() (dfvfs.vfs.tsk_file_entry.TSKFileEntry method), 238	GetFormatSpecification() (dfvfs.analyzer.ntfs_analyzer_helper.NTFSEntry method), 38
GetFileObjectByPathSpec() (dfvfs.vfs.file_system.FileSystem method), 208	GetFormatSpecification() (dfvfs.analyzer.phdi_analyzer_helper.PHDIAnalyzerHelper method), 39
GetFileSystem() (dfvfs.resolver.context.Context method), 147	GetFormatSpecification() (dfvfs.analyzer.qcow_analyzer_helper.QCOWAnalyzerHelper method), 39
GetFileSystem() (dfvfs.vfs.file_entry.FileEntry method), 203	GetFormatSpecification() (dfvfs.analyzer.tar_analyzer_helper.TARAnalyzerHelper method), 42
GetFileSystemTypeIndicators() (dfvfs.analyzer.analyzer.Analyzer method), 30	
GetFormatSpecification() (dfvfs.analyzer.analyzer_helper.AnalyzerHelper method), 31	
GetFormatSpecification() (dfvfs.analyzer.apfs_analyzer_helper.APFSAnalyzerHelper method), 32	

GetFormatSpecification() (dfvfs.analyzer.tsk_analyzer_helper.TSKAnalyzerHelper method), 42	GetLinkedFileEntry() (dfvfs.vfs.file_entry.FileEntry method), 203
GetFormatSpecification() (dfvfs.analyzer.vhdi_analyzer_helper.VHDIAnalyzerHelper method), 43	GetLinkedFileEntry() (dfvfs.vfs.hfs_file_entry.HFSFileEntry method), 215
GetFormatSpecification() (dfvfs.analyzer.vmdk_analyzer_helper.VMDKAnalyzerHelper method), 43	GetLinkedFileEntry() (dfvfs.vfs.ntfs_file_entry.NTFSFileEntry method), 224
GetFormatSpecification() (dfvfs.analyzer.vshadow_analyzer_helper.VShadowAnalyzerHelper method), 44	GetLinkedFileEntry() (dfvfs.vfs.os_file_entry.OSFileEntry method), 224
GetFormatSpecification() (dfvfs.analyzer.xfs_analyzer_helper.XFSAnalyzerHelper method), 44	GetLinkedFileEntry() (dfvfs.vfs.tsk_file_entry.TSKFileEntry method), 238
GetFormatSpecification() (dfvfs.analyzer.xz_analyzer_helper.XZAnalyzerHelper method), 45	GetLinkedFileEntry() (dfvfs.vfs.xfs_file_entry.XFSFileEntry method), 249
GetFormatSpecification() (dfvfs.analyzer.zip_analyzer_helper.ZipAnalyzerHelper method), 45	GetLUKSDEVolume() (dfvfs.vfs.luksde_file_system.LUKSDEFileSystem method), 219
GetFsInfo() (dfvfs.vfs.tsk_file_system.TSKFileSystem method), 241	GetLVMLogicalVolume() (dfvfs.vfs.lvm_file_entry.LVMFileEntry method), 219
GetFsType() (dfvfs.vfs.tsk_file_system.TSKFileSystem method), 241	GetLVMLogicalVolumeByPathSpec() (dfvfs.vfs.lvm_file_system.LVMFileSystem method), 220
GetFVDELogicalVolume() (dfvfs.vfs.cs_file_entry.CSFileEntry method), 185	GetLVMVolumeGroup() (dfvfs.vfs.lvm_file_system.LVMFileSystem method), 220
GetFVDELogicalVolumeByPathSpec() (dfvfs.vfs.cs_file_system.CSFileSystem method), 186	GetLVMVolumeIdentifiers() (dfvfs.helpers.command_line.CLIVolumeScannerMediator method), 93
GetFVDEVVolumeGroup() (dfvfs.vfs.cs_file_system.CSFileSystem method), 187	GetLVMVolumeIdentifiers() (dfvfs.helpers.volume_scanner.VolumeScannerMediator method), 109
GetGPTPartition() (dfvfs.vfs.gpt_file_entry.GPTFileEntry method), 210	GetMountPoint() (dfvfs.mount.manager.MountPointManager class method), 128
GetGPTPartitionByPathSpec() (dfvfs.vfs.gpt_file_system.GPTFileSystem method), 211	GetMountPoint() (dfvfs.resolver.context.Context method), 147
GetGPTVolume() (dfvfs.vfs.gpt_file_system.GPTFileSystem method), 211	GetNTFSFileEntry() (dfvfs.vfs.ntfs_file_entry.NTFSFileEntry method), 224
GetHelper() (dfvfs.resolver_helpers.manager.ResolverHelperManager class method), 161	GetNTFSFileEntryByPathSpec() (dfvfs.vfs.ntfs_file_system.NTFSFileSystem method), 226
GetHFSFileEntry() (dfvfs.vfs.hfs_file_entry.HFSFileEntry method), 215	GetNumberOfRows() (dfvfs.file_io.sqlite_blob_file_io.SQLiteBlobFile method), 83
GetHFSFileEntryByPathSpec() (dfvfs.vfs.hfs_file_system.HFSFileSystem method), 217	GetNumberOfRows() (dfvfs.lib.sqlite_database.SQLiteDatabaseFile method), 124
GetLinkedFileEntry() (dfvfs.vfs.apfs_file_entry.APFSFileEntry method), 177	GetNumberOfRows() (dfvfs.vfs.sqlite_blob_file_entry.SQLiteBlobFileEntry method), 232
GetLinkedFileEntry() (dfvfs.vfs.ext_file_entry.EXTFileEntry method), 193	GetParentFileEntry() (dfvfs.vfs.apfs_container_file_entry.APFSContainerFileEntry method), 174
	GetParentFileEntry() (dfvfs.vfs.apfs_file_entry.APFSFileEntry method), 174

- method), 177
- GetParentFileEntry()
(dfvfs.vfs.cpio_file_entry.CPIOFileEntry method), 183
- GetParentFileEntry()
(dfvfs.vfs.cs_file_entry.CSFileEntry method), 185
- GetParentFileEntry()
(dfvfs.vfs.ext_file_entry.EXTFileEntry method), 193
- GetParentFileEntry()
(dfvfs.vfs.fake_file_entry.FakeFileEntry method), 196
- GetParentFileEntry()
(dfvfs.vfs.fat_file_entry.FATFileEntry method), 199
- GetParentFileEntry() (dfvfs.vfs.file_entry.FileEntry method), 203
- GetParentFileEntry()
(dfvfs.vfs.gpt_file_entry.GPTFileEntry method), 210
- GetParentFileEntry()
(dfvfs.vfs.hfs_file_entry.HFSFileEntry method), 215
- GetParentFileEntry()
(dfvfs.vfs.lvm_file_entry.LVMFileEntry method), 219
- GetParentFileEntry()
(dfvfs.vfs.ntfs_file_entry.NTFSFileEntry method), 225
- GetParentFileEntry()
(dfvfs.vfs.os_file_entry.OSFileEntry method), 229
- GetParentFileEntry()
(dfvfs.vfs.sqlite_blob_file_entry.SQLiteBlobFileEntry method), 232
- GetParentFileEntry()
(dfvfs.vfs.tar_file_entry.TARFileEntry method), 234
- GetParentFileEntry()
(dfvfs.vfs.tsk_file_entry.TSKFileEntry method), 238
- GetParentFileEntry()
(dfvfs.vfs.tsk_partition_file_entry.TSKPartitionFileEntry method), 243
- GetParentFileEntry()
(dfvfs.vfs.vshadow_file_entry.VShadowFileEntry method), 245
- GetParentFileEntry()
(dfvfs.vfs.xfs_file_entry.XFSFileEntry method), 249
- GetParentFileEntry()
(dfvfs.vfs.zip_file_entry.ZipFileEntry method), 251
- GetPartitionIdentifiers()
(dfvfs.helpers.command_line.CLIVolumeScannerMediator method), 93
- GetPartitionIdentifiers()
(dfvfs.helpers.volume_scanner.VolumeScannerMediator method), 109
- GetPaths() (dfvfs.vfs.fake_file_system.FakeFileSystem method), 198
- GetPathSegmentAndSuffix()
(dfvfs.vfs.file_system.FileSystem method), 208
- GetProperties() (dfvfs.path.factory.Factory class method), 134
- GetRelativePath() (dfvfs.helpers.file_system_searcher.FileSystemSearcher method), 97
- GetRootFileEntry() (dfvfs.vfs.apfs_container_file_system.APFSContainerFileEntry method), 176
- GetRootFileEntry() (dfvfs.vfs.apfs_file_system.APFSFileEntry method), 179
- GetRootFileEntry() (dfvfs.vfs.bde_file_system.BDEFileEntry method), 182
- GetRootFileEntry() (dfvfs.vfs.compressed_stream_file_system.CompressedStreamFileEntry method), 183
- GetRootFileEntry() (dfvfs.vfs.cpio_file_system.CPIOFileEntry method), 185
- GetRootFileEntry() (dfvfs.vfs.cs_file_system.CSFileEntry method), 187
- GetRootFileEntry() (dfvfs.vfs.data_range_file_system.DataRangeFileEntry method), 188
- GetRootFileEntry() (dfvfs.vfs.encoded_stream_file_system.EncodedStreamFileEntry method), 190
- GetRootFileEntry() (dfvfs.vfs.encrypted_stream_file_system.EncryptedStreamFileEntry method), 191
- GetRootFileEntry() (dfvfs.vfs.ext_file_system.EXTFileEntry method), 195
- GetRootFileEntry() (dfvfs.vfs.fake_file_system.FakeFileEntry method), 198
- GetRootFileEntry() (dfvfs.vfs.fat_file_system.FATFileEntry method), 201
- GetRootFileEntry() (dfvfs.vfs.file_system.FileSystem method), 209
- GetRootFileEntry() (dfvfs.vfs.gpt_file_system.GPTFileEntry method), 211
- GetRootFileEntry() (dfvfs.vfs.gzip_file_system.GzipFileEntry method), 212
- GetRootFileEntry() (dfvfs.vfs.hfs_file_system.HFSFileEntry method), 217
- GetRootFileEntry() (dfvfs.vfs.luksde_file_system.LUKSDEFileEntry method), 219
- GetRootFileEntry() (dfvfs.vfs.lvm_file_system.LVMFileEntry method), 221
- GetRootFileEntry() (dfvfs.vfs.ntfs_file_system.NTFSFileEntry method), 227
- GetRootFileEntry() (dfvfs.vfs.os_file_system.OSFileEntry method), 229

method), 230

GetRootFileEntry() (dfvfs.vfs.root_only_file_system.RootOnlyFileSystem method), 244

method), 231

GetRootFileEntry() (dfvfs.vfs.sqlite_blob_file_system.SQLiteBlobFileSystem method), 243

method), 233

GetRootFileEntry() (dfvfs.vfs.tar_file_system.TARFileSystem dfvfs.lib.tsk_partition), 126

method), 235

GetRootFileEntry() (dfvfs.vfs.tsk_file_system.TSKFileSystem (dfvfs.helpers.source_scanner.SourceScannerContext method), 104

method), 242

GetRootFileEntry() (dfvfs.vfs.tsk_partition_file_system.TSKPartitionFileSystem (dfvfs.helpers.source_scanner.SourceScanNode method), 104

method), 244

GetRootFileEntry() (dfvfs.vfs.vshadow_file_system.VShadowFileSystem method), 100

method), 246

GetRootFileEntry() (dfvfs.vfs.xfs_file_system.XFSFileSystem (dfvfs.volume.volume_system.VolumeSystem method), 257

method), 250

GetRootFileEntry() (dfvfs.vfs.zip_file_system.ZipFileSystem (dfvfs.volume.volume_system.VolumeSystem method), 257

method), 252

GetRootInode() (dfvfs.vfs.tsk_file_system.TSKFileSystem (dfvfs.helpers.source_scanner.SourceScannerContext method), 102

method), 242

GetRootScanNode() (dfvfs.helpers.source_scanner.SourceScannerContext method), 104

method), 104

GetScanNode() (dfvfs.helpers.source_scanner.SourceScannerContext (dfvfs.helpers.volume_scanner.VolumeScannerMediator method), 109

method), 104

GetSectionByIndex() (dfvfs.volume.volume_system.VolumeSystem (dfvfs.helpers.volume_scanner.VolumeScannerMediator method), 110

method), 257

GetSecurityDescriptor() (dfvfs.vfs.ntfs_file_entry.NTFSFileEntry (dfvfs.analyzer.analyzer.Analyzer class method), 30

method), 225

GetSpecificationBySignature() (dfvfs.analyzer.specification.FormatSpecificationStore method), 41

method), 41

GetStatAttribute() (dfvfs.vfs.file_entry.FileEntry (dfvfs.vfs.vshadow_file_system.VShadowFileSystem method), 246

method), 203

GetStorageMediaImageTypeIndicators() (dfvfs.analyzer.analyzer.Analyzer class method), 30

method), 30

GetSubFileEntryByName() (dfvfs.vfs.file_entry.FileEntry method), 203

method), 203

GetSubNodeByLocation() (dfvfs.helpers.source_scanner.SourceScanNode (dfvfs.helpers.volume_scanner.VolumeScannerMediator method), 109

method), 100

GetTARFile() (dfvfs.vfs.tar_file_system.TARFileSystem (dfvfs.helpers.windows_path_resolver.WindowsPathResolver method), 112

method), 235

GetTARInfo() (dfvfs.vfs.tar_file_entry.TARFileEntry (dfvfs.vfs.xfs_file_entry.XFSFileEntry method), 249

method), 234

GetTARInfoByPathSpec() (dfvfs.vfs.tar_file_system.TARFileSystem (dfvfs.vfs.xfs_file_system.XFSFileSystem method), 250

method), 235

GetTSKFile() (dfvfs.vfs.tsk_file_entry.TSKFileEntry (dfvfs.vfs.zip_file_system.ZipFileSystem method), 253

method), 239

GetTSKFileByPathSpec() (dfvfs.vfs.tsk_file_system.TSKFileSystem (dfvfs.vfs.zip_file_entry.ZipFileEntry method), 251

method), 242

GetTSKVolume() (dfvfs.vfs.tsk_partition_file_system.TSKPartitionFileSystem (dfvfs.vfs.zip_file_system.ZipFileSystem method), 253

method), 244

GetTSKVsPart() (dfvfs.vfs.tsk_partition_file_entry.TSKPartitionFileEntry (dfvfs.vfs.zip_file_system.ZipFileSystem method), 253

method), 243

GetTSKVsPartByPathSpec() (in module (dfvfs.volume.volume_system.VolumeSystem method), 257

dfvfs.lib.tsk_partition), 126

GetUnscannedScanNode() (dfvfs.helpers.source_scanner.SourceScannerContext (dfvfs.helpers.volume_scanner.VolumeScannerMediator method), 109

method), 104

GetVolumeByIdentifier() (dfvfs.volume.volume_system.VolumeSystem (dfvfs.helpers.volume_scanner.VolumeScannerMediator method), 110

method), 257

GetVolumeByIndex() (dfvfs.volume.volume_system.VolumeSystem (dfvfs.helpers.volume_scanner.VolumeScannerMediator method), 110

method), 257

GetVolumeIdentifiers() (dfvfs.helpers.source_scanner.SourceScannerContext (dfvfs.helpers.volume_scanner.VolumeScannerMediator method), 109

method), 102

GetVolumeIdentifiers() (dfvfs.helpers.volume_scanner.VolumeScannerMediator (dfvfs.analyzer.analyzer.Analyzer class method), 30

method), 110

GetVolumeSnapshotIdentifiers() (dfvfs.helpers.volume_scanner.VolumeScannerMediator (dfvfs.analyzer.analyzer.Analyzer class method), 30

method), 110

GetVolumeSystemTypeIndicators() (dfvfs.analyzer.analyzer.Analyzer class method), 30

method), 30

GetVShadowStore() (dfvfs.vfs.vshadow_file_entry.VShadowFileEntry (dfvfs.vfs.vshadow_file_system.VShadowFileSystem method), 246

method), 245

GetVShadowStoreByPathSpec() (dfvfs.vfs.vshadow_file_system.VShadowFileSystem (dfvfs.vfs.vshadow_file_system.VShadowFileSystem method), 246

method), 246

GetVShadowVolume() (dfvfs.vfs.vshadow_file_system.VShadowFileSystem (dfvfs.vfs.vshadow_file_system.VShadowFileSystem method), 246

method), 246

GetVSSStoreIdentifiers() (dfvfs.helpers.command_line.CLIVolumeScannerMediator (dfvfs.helpers.volume_scanner.VolumeScannerMediator method), 109

method), 93

GetVSSStoreIdentifiers() (dfvfs.helpers.volume_scanner.VolumeScannerMediator (dfvfs.helpers.volume_scanner.VolumeScannerMediator method), 109

method), 109

GetWindowsPath() (dfvfs.helpers.windows_path_resolver.WindowsPathResolver (dfvfs.vfs.xfs_file_entry.XFSFileEntry method), 249

method), 112

GetXFSFileEntry() (dfvfs.vfs.xfs_file_entry.XFSFileEntry (dfvfs.vfs.xfs_file_system.XFSFileSystem method), 250

method), 249

GetXFSFileEntryByPathSpec() (dfvfs.vfs.xfs_file_system.XFSFileSystem (dfvfs.vfs.zip_file_system.ZipFileSystem method), 253

method), 250

GetZipFile() (dfvfs.vfs.zip_file_system.ZipFileSystem (dfvfs.vfs.zip_file_entry.ZipFileEntry method), 251

method), 253

GetZipInfo() (dfvfs.vfs.zip_file_entry.ZipFileEntry (dfvfs.vfs.zip_file_entry.ZipFileEntry method), 251

method), 251

GetZipInfoByPathSpec() (dfvfs.vfs.zip_file_entry.ZipFileEntry (dfvfs.vfs.zip_file_entry.ZipFileEntry method), 251

method), 251

- (*dfvfs.vfs.zip_file_system.ZipFileSystem* method), 253
- Glob2Regex() (in module *dfvfs.lib.glob2regex*), 119
- GPTAnalyzerHelper (class in *dfvfs.analyzer.gpt_analyzer_helper*), 36
- GPTDirectory (class in *dfvfs.vfs.gpt_directory*), 210
- GPTFile (class in *dfvfs.file_io.gpt_file_io*), 75
- GPTFileEntry (class in *dfvfs.vfs.gpt_file_entry*), 210
- GPTFileSystem (class in *dfvfs.vfs.gpt_file_system*), 211
- GPTPathSpec (class in *dfvfs.path.gpt_path_spec*), 136
- GPTPathSpecGetEntryIndex() (in module *dfvfs.lib.gpt_helper*), 119
- GPTResolverHelper (class in *dfvfs.resolver_helpers.gpt_resolver_helper*), 157
- GPTVolume (class in *dfvfs.volume.gpt_volume_system*), 255
- GPTVolumeSystem (class in *dfvfs.volume.gpt_volume_system*), 255
- group_idenfifier (*dfvfs.lib.cpio.CPIOArchiveFileEntry* attribute), 115
- group_idenfifier (*dfvfs.vfs.attribute.StatAttribute* attribute), 179
- GzipAnalyzerHelper (class in *dfvfs.analyzer.gzip_analyzer_helper*), 36
- GzipCompressedStream (class in *dfvfs.lib.gzipfile*), 119
- GzipFile (class in *dfvfs.file_io.gzip_file_io*), 76
- GzipFileEntry (class in *dfvfs.vfs.gzip_file_entry*), 212
- GzipFileSystem (class in *dfvfs.vfs.gzip_file_system*), 212
- GzipMember (class in *dfvfs.lib.gzipfile*), 121
- GzipPathSpec (class in *dfvfs.path.gzip_path_spec*), 136
- GzipResolverHelper (class in *dfvfs.resolver_helpers.gzip_resolver_helper*), 158
- ## H
- HasColumn() (*dfvfs.lib.sqlite_database.SQLiteDatabaseFileEntry* method), 124
- HasDataStream() (*dfvfs.vfs.file_entry.FileEntry* method), 203
- HasExternalData() (*dfvfs.vfs.file_entry.FileEntry* method), 204
- HasExternalData() (*dfvfs.vfs.vshadow_file_entry.VShadowFileEntry* method), 245
- HasExternalData() (*dfvfs.volume.volume_system.VolumeSystem* method), 256
- HasExternalData() (*dfvfs.volume.vshadow_volume_system.VShadowVolumeSystem* method), 258
- HasFileSystemScanNodes() (*dfvfs.helpers.source_scanner.SourceScannerContext* method), 104
- HasLocation() (*dfvfs.helpers.file_system_searcher.FindSpace* method), 99
- HasLockedScanNodes() (*dfvfs.helpers.source_scanner.SourceScannerContext* method), 104
- HasParent() (*dfvfs.path.path_spec.PathSpec* method), 140
- HasScanNode() (*dfvfs.helpers.source_scanner.SourceScannerContext* method), 105
- HasTable() (*dfvfs.lib.sqlite_database.SQLiteDatabaseFile* method), 124
- HFSAnalyzerHelper (class in *dfvfs.analyzer.hfs_analyzer_helper*), 37
- HFSDataStream (class in *dfvfs.vfs.hfs_data_stream*), 214
- HFSDirectory (class in *dfvfs.vfs.hfs_directory*), 215
- HFSExtendedAttribute (class in *dfvfs.vfs.hfs_attribute*), 213
- HFSFile (class in *dfvfs.file_io.hfs_file_io*), 77
- HFSFileEntry (class in *dfvfs.vfs.hfs_file_entry*), 215
- HFSFileSystem (class in *dfvfs.vfs.hfs_file_system*), 217
- HFSPathSpec (class in *dfvfs.path.hfs_path_spec*), 137
- HFSResolverHelper (class in *dfvfs.resolver_helpers.hfs_resolver_helper*), 159
- ## I
- identififier (*dfvfs.analyzer.specification.Signature* attribute), 41
- identififier (*dfvfs.path.apfs_path_spec.APFSPathSpec* attribute), 129
- identififier (*dfvfs.path.fat_path_spec.FATPathSpec* attribute), 135
- identififier (*dfvfs.path.hfs_path_spec.HFSPathSpec* attribute), 137
- identififier (*dfvfs.path.mount_path_spec.MountPathSpec* attribute), 139
- initialization_vector (*dfvfs.path.encrypted_stream_path_spec.EncryptedStreamPathSpec* attribute), 133
- inode (*dfvfs.path.ext_path_spec.EXTPathSpec* attribute), 133
- inode (*dfvfs.path.tsk_path_spec.TSKPathSpec* attribute), 144
- inode (*dfvfs.path.xfs_path_spec.XFSPathSpec* attribute), 146
- inode_number (*dfvfs.lib.cpio.CPIOArchiveFileEntry* attribute), 115
- inode_number (*dfvfs.vfs.attribute.StatAttribute* attribute), 180
- is_locked (*dfvfs.file_io.bde_file_io.BDEFile* property), 60
- is_locked (*dfvfs.file_io.cs_file_io.CSFile* property), 63
- is_locked (*dfvfs.file_io.luksde_file_io.LUKSDEFile* property), 78
- is_allocated() (*dfvfs.vfs.file_entry.FileEntry* method), 204

- IsAllocated() (*dfvfs.vfs.ntfs_file_entry.NTFSFileEntry* method), 225
 IsAllocated() (*dfvfs.vfs.tsk_file_entry.TSKFileEntry* method), 239
 IsAllocated() (*dfvfs.vfs.tsk_partition_file_entry.TSKPartitionFileEntry* method), 243
 IsDefault() (*dfvfs.vfs.data_stream.DataStream* method), 188
 IsDevice() (*dfvfs.vfs.file_entry.FileEntry* method), 204
 IsDirectory() (*dfvfs.vfs.file_entry.FileEntry* method), 204
 IsEnabled() (*dfvfs.analyzer.analyzer_helper.AnalyzerHelper* method), 31
 IsEnabled() (*dfvfs.analyzer.ext_analyzer_helper.EXTAnalyzerHelper* method), 35
 IsEnabled() (*dfvfs.analyzer.fat_analyzer_helper.FATAnalyzerHelper* method), 35
 IsEnabled() (*dfvfs.analyzer.gpt_analyzer_helper.GPTAnalyzerHelper* method), 36
 IsEnabled() (*dfvfs.analyzer.hfs_analyzer_helper.HFSAnalyzerHelper* method), 37
 IsEnabled() (*dfvfs.analyzer.ntfs_analyzer_helper.NTFSAnalyzerHelper* method), 39
 IsExt() (*dfvfs.vfs.tsk_file_system.TSKFileSystem* method), 242
 IsFile() (*dfvfs.vfs.file_entry.FileEntry* method), 204
 IsFileSystem() (*dfvfs.helpers.source_scanner.SourceScannerContext* method), 101
 IsFileSystem() (*dfvfs.path.path_spec.PathSpec* method), 141
 IsHFS() (*dfvfs.vfs.tsk_file_system.TSKFileSystem* method), 242
 IsLastLocationSegment() (*dfvfs.helpers.file_system_searcher.FindSpec* method), 99
 IsLink() (*dfvfs.vfs.file_entry.FileEntry* method), 204
 IsLocked() (*dfvfs.vfs.apfs_container_file_entry.APFSContainerFileEntry* method), 174
 IsLocked() (*dfvfs.vfs.bde_file_entry.BDEFileEntry* method), 181
 IsLocked() (*dfvfs.vfs.cs_file_entry.CSFileEntry* method), 186
 IsLocked() (*dfvfs.vfs.file_entry.FileEntry* method), 204
 IsLocked() (*dfvfs.vfs.luksde_file_entry.LUKSDEFileEntry* method), 218
 IsLockedScanNode() (*dfvfs.helpers.source_scanner.SourceScannerContext* method), 105
 IsNTFS() (*dfvfs.vfs.tsk_file_system.TSKFileSystem* method), 242
 IsPipe() (*dfvfs.vfs.file_entry.FileEntry* method), 205
 IsRoot() (*dfvfs.vfs.file_entry.FileEntry* method), 205
 IsSocket() (*dfvfs.vfs.file_entry.FileEntry* method), 205
 IsSourceTypeDirectory() (*dfvfs.helpers.source_scanner.SourceScannerContext* method), 105
 IsSourceTypeFile() (*dfvfs.helpers.source_scanner.SourceScannerContext* method), 105
 IsSystemLevel() (*dfvfs.helpers.source_scanner.SourceScanNode* method), 101
 IsSystemLevel() (*dfvfs.path.path_spec.PathSpec* method), 141
 IsSystemLevelTypeIndicator() (*dfvfs.path.factory.Factory* class method), 134
 IsVirtual() (*dfvfs.vfs.file_entry.FileEntry* method), 205
 IsVolumeSystem() (*dfvfs.helpers.source_scanner.SourceScanNode* method), 101
 IsVolumeSystem() (*dfvfs.path.path_spec.PathSpec* method), 141
 IsVolumeSystemRoot() (*dfvfs.helpers.source_scanner.SourceScanNode* method), 101
 IsVolumeSystemRoot() (*dfvfs.path.path_spec.PathSpec* method), 141
- ## J
- JoinPath() (*dfvfs.vfs.file_system.FileSystem* method), 209
 JoinPath() (*dfvfs.vfs.os_file_system.OSFileSystem* method), 230
 JsonPathSpecSerializer (class in *dfvfs.serializer.json_serializer*), 171
- ## K
- key (*dfvfs.path.encrypted_stream_path_spec.EncryptedStreamPathSpec* attribute), 133
 key_chain (*dfvfs.resolver.resolver.Resolver* attribute), 149
 KeyChain (class in *dfvfs.credentials.keychain*), 50
- ## L
- link (*dfvfs.vfs.file_entry.FileEntry* property), 206
 location (*dfvfs.path.apfs_container_path_spec.APFSContainerPathSpec* attribute), 128
 location (*dfvfs.path.apfs_path_spec.APFSPathSpec* attribute), 129
 location (*dfvfs.path.cs_path_spec.CSPathSpec* attribute), 131
 location (*dfvfs.path.ext_path_spec.EXTPathSpec* attribute), 133
 location (*dfvfs.path.fat_path_spec.FATPathSpec* attribute), 135
 location (*dfvfs.path.gpt_path_spec.GPTPathSpec* attribute), 136
 location (*dfvfs.path.hfs_path_spec.HFSPathSpec* attribute), 137

location (*dfvfs.path.location_path_spec.LocationPathSpec* attribute), 137
 location (*dfvfs.path.lvm_path_spec.LVMPathSpec* attribute), 138
 location (*dfvfs.path.ntfs_path_spec.NTFSPathSpec* attribute), 139
 location (*dfvfs.path.tsk_partition_path_spec.TSKPartitionPathSpec* attribute), 143
 location (*dfvfs.path.tsk_path_spec.TSKPathSpec* attribute), 144
 location (*dfvfs.path.vshadow_path_spec.VShadowPathSpec* attribute), 145
 location (*dfvfs.path.xfs_path_spec.XFSPathSpec* attribute), 146
 LOCATION_ROOT (*dfvfs.vfs.fat_file_system.FATFilesystem* attribute), 201
 LOCATION_ROOT (*dfvfs.vfs.file_system.FileSystem* attribute), 209
 LOCATION_ROOT (*dfvfs.vfs.ntfs_file_system.NTFSTFilesystem* attribute), 227
 LocationPathSpec (class in *dfvfs.path.location_path_spec*), 137
 locked_scan_nodes (*dfvfs.helpers.source_scanner.SourceScannerContext* property), 106
 LockScanNode() (*dfvfs.helpers.source_scanner.SourceScannerContext* method), 105
 LUKSDEAnalyzerHelper (class in *dfvfs.analyzer.luksde_analyzer_helper*), 37
 LUKSDECredentials (class in *dfvfs.credentials.luksde_credentials*), 51
 LUKSDEFile (class in *dfvfs.file_io.luksde_file_io*), 78
 LUKSDEFileEntry (class in *dfvfs.vfs.luksde_file_entry*), 218
 LUKSDEFilesystem (class in *dfvfs.vfs.luksde_file_system*), 218
 LUKSDEOpenVolume() (in module *dfvfs.lib.luksde_helper*), 122
 LUKSDEPathSpec (class in *dfvfs.path.luksde_path_spec*), 138
 LUKSDEResolverHelper (class in *dfvfs.resolver_helpers.luksde_resolver_helper*), 159
 LUKSDEUnlockVolume() (in module *dfvfs.lib.luksde_helper*), 122
 LVMAnalyzerHelper (class in *dfvfs.analyzer.lvm_analyzer_helper*), 38
 LVMDirectory (class in *dfvfs.vfs.lvm_directory*), 219
 LVMDFile (class in *dfvfs.file_io.lvm_file_io*), 78
 LVMDFileEntry (class in *dfvfs.vfs.lvm_file_entry*), 219
 LVMDFilesystem (class in *dfvfs.vfs.lvm_file_system*), 220
 LVMPathSpec (class in *dfvfs.path.lvm_path_spec*), 138
 LVMPathSpecGetVolumeIndex() (in module *dfvfs.lib.lvm_helper*), 123
 LVMResolverHelper (class in *dfvfs.resolver_helpers.lvm_resolver_helper*), 160
 LVMVolume (class in *dfvfs.volume.lvm_volume_system*), 255
 LVMVolumeSystem (class in *dfvfs.volume.lvm_volume_system*), 255
 LZMACompressor (class in *dfvfs.compression.xz_decompressor*), 48

M

member_end_offset (*dfvfs.lib.zipfile.GzipMember* attribute), 121
 member_start_offset (*dfvfs.lib.zipfile.GzipMember* attribute), 121
 members (*dfvfs.lib.zipfile.GzipCompressedStream* property), 120
 mft_attribute (*dfvfs.path.ntfs_path_spec.NTFSPathSpec* attribute), 139
 mft_entry (*dfvfs.path.ntfs_path_spec.NTFSPathSpec* attribute), 140
 MFT_ENTRY_ROOT_DIRECTORY (*dfvfs.vfs.ntfs_file_system.NTFSTFilesystem* attribute), 227
 mode (*dfvfs.lib.cpio.CPIOArchiveFileEntry* attribute), 115
 mode (*dfvfs.vfs.attribute.StatAttribute* attribute), 180
 MODIANalyzerHelper (class in *dfvfs.analyzer.modi_analyzer_helper*), 38
 modification_time (*dfvfs.lib.cpio.CPIOArchiveFileEntry* attribute), 116
 modification_time (*dfvfs.vfs.apfs_file_entry.APFSFileEntry* property), 177
 modification_time (*dfvfs.vfs.cpio_file_entry.CPIOFileEntry* property), 183
 modification_time (*dfvfs.vfs.ext_file_entry.EXTFileEntry* property), 194
 modification_time (*dfvfs.vfs.fake_file_entry.FakeFileEntry* property), 197
 modification_time (*dfvfs.vfs.fat_file_entry.FATFileEntry* property), 200
 modification_time (*dfvfs.vfs.file_entry.FileEntry* property), 206
 modification_time (*dfvfs.vfs.zip_file_entry.GzipFileEntry* property), 212
 modification_time (*dfvfs.vfs.hfs_file_entry.HFSFileEntry* property), 216
 modification_time (*dfvfs.vfs.ntfs_attribute.FileNameNTFSAttribute* property), 221
 modification_time (*dfvfs.vfs.ntfs_attribute.StandardInformationNTFSAttribute* property), 223
 modification_time (*dfvfs.vfs.ntfs_file_entry.NTFSTFileEntry* property), 225
 modification_time (*dfvfs.vfs.os_file_entry.OSFileEntry* property), 229

modification_time (*dfvfs.vfs.tar_file_entry.TARFileEntry* property), 234
 modification_time (*dfvfs.vfs.tsk_file_entry.TSKFileEntry* property), 239
 modification_time (*dfvfs.vfs.xfs_file_entry.XFSFileEntry* property), 249
 modification_time (*dfvfs.vfs.zip_file_entry.ZipFileEntry* property), 251
 modification_times (*dfvfs.file_io.gzip_file_io.GzipFile* property), 76
 MODIFile (class in *dfvfs.file_io.modi_file_io*), 79
 MODIPathSpec (class in *dfvfs.path.modi_path_spec*), 139
 MODIResolverHelper (class in *dfvfs.resolver_helpers.modi_resolver_helper*), 161
 module
 dfvfs, 259
 dfvfs.analyzer, 45
 dfvfs.analyzer.analyzer, 29
 dfvfs.analyzer.analyzer_helper, 31
 dfvfs.analyzer.apfs_analyzer_helper, 32
 dfvfs.analyzer.apfs_container_analyzer_helper, 32
 dfvfs.analyzer.bde_analyzer_helper, 32
 dfvfs.analyzer.bzip2_analyzer_helper, 33
 dfvfs.analyzer.cpio_analyzer_helper, 33
 dfvfs.analyzer.cs_analyzer_helper, 34
 dfvfs.analyzer.ewf_analyzer_helper, 34
 dfvfs.analyzer.ext_analyzer_helper, 34
 dfvfs.analyzer.fat_analyzer_helper, 35
 dfvfs.analyzer.gpt_analyzer_helper, 36
 dfvfs.analyzer.gzip_analyzer_helper, 36
 dfvfs.analyzer.hfs_analyzer_helper, 37
 dfvfs.analyzer.luksde_analyzer_helper, 37
 dfvfs.analyzer.lvm_analyzer_helper, 38
 dfvfs.analyzer.modi_analyzer_helper, 38
 dfvfs.analyzer.ntfs_analyzer_helper, 38
 dfvfs.analyzer.phdi_analyzer_helper, 39
 dfvfs.analyzer.qcow_analyzer_helper, 39
 dfvfs.analyzer.specification, 40
 dfvfs.analyzer.tar_analyzer_helper, 42
 dfvfs.analyzer.tsk_analyzer_helper, 42
 dfvfs.analyzer.tsk_partition_analyzer_helper, 42
 dfvfs.analyzer.vhdi_analyzer_helper, 43
 dfvfs.analyzer.vmdk_analyzer_helper, 43
 dfvfs.analyzer.vshadow_analyzer_helper, 44
 dfvfs.analyzer.xfs_analyzer_helper, 44
 dfvfs.analyzer.xz_analyzer_helper, 45
 dfvfs.analyzer.zip_analyzer_helper, 45
 dfvfs.compression, 49
 dfvfs.compression.bzip2_decompressor, 46
 dfvfs.compression.decompressor, 46
 dfvfs.compression.manager, 47
 dfvfs.compression.xz_decompressor, 48
 dfvfs.compression.zlib_decompressor, 48
 dfvfs.credentials, 52
 dfvfs.credentials.apfs_credentials, 49
 dfvfs.credentials.bde_credentials, 49
 dfvfs.credentials.credentials, 49
 dfvfs.credentials.cs_credentials, 50
 dfvfs.credentials.encrypted_stream_credentials, 50
 dfvfs.credentials.keychain, 50
 dfvfs.credentials.luksde_credentials, 51
 dfvfs.credentials.manager, 52
 dfvfs.encoding, 55
 dfvfs.encoding.base16_decoder, 53
 dfvfs.encoding.base32_decoder, 53
 dfvfs.encoding.base64_decoder, 54
 dfvfs.encoding.decoder, 54
 dfvfs.encoding.manager, 54
 dfvfs.encryption, 58
 dfvfs.encryption.aes_decrypter, 55
 dfvfs.encryption.blowfish_decrypter, 56
 dfvfs.encryption.decrypter, 56
 dfvfs.encryption.des3_decrypter, 57
 dfvfs.encryption.manager, 57
 dfvfs.encryption.rc4_decrypter, 58
 dfvfs.file_io, 92
 dfvfs.file_io.apfs_file_io, 58
 dfvfs.file_io.bde_file_io, 60
 dfvfs.file_io.compressed_stream_io, 60
 dfvfs.file_io.cpio_file_io, 61
 dfvfs.file_io.cs_file_io, 62
 dfvfs.file_io.data_range_io, 64
 dfvfs.file_io.encoded_stream_io, 65
 dfvfs.file_io.encrypted_stream_io, 66
 dfvfs.file_io.ewf_file_io, 68
 dfvfs.file_io.ext_file_io, 68
 dfvfs.file_io.fake_file_io, 69
 dfvfs.file_io.fat_file_io, 70
 dfvfs.file_io.file_io, 72
 dfvfs.file_io.file_object_io, 73
 dfvfs.file_io.gpt_file_io, 75
 dfvfs.file_io.gzip_file_io, 76
 dfvfs.file_io.hfs_file_io, 77
 dfvfs.file_io.luksde_file_io, 78
 dfvfs.file_io.lvm_file_io, 78
 dfvfs.file_io.modi_file_io, 79
 dfvfs.file_io.ntfs_file_io, 80
 dfvfs.file_io.os_file_io, 81
 dfvfs.file_io.phdi_file_io, 82
 dfvfs.file_io.qcow_file_io, 82
 dfvfs.file_io.raw_file_io, 83
 dfvfs.file_io.sqlite_blob_file_io, 83
 dfvfs.file_io.tar_file_io, 85

dfvfs.file_io.tsk_file_io, 86
 dfvfs.file_io.tsk_partition_file_io, 87
 dfvfs.file_io.vhdi_file_io, 87
 dfvfs.file_io.vmdk_file_io, 88
 dfvfs.file_io.vshadow_file_io, 88
 dfvfs.file_io.xfs_file_io, 89
 dfvfs.file_io.zip_file_io, 90
 dfvfs.helpers, 113
 dfvfs.helpers.command_line, 92
 dfvfs.helpers.data_slice, 95
 dfvfs.helpers.fake_file_system_builder, 96
 dfvfs.helpers.file_system_searcher, 97
 dfvfs.helpers.source_scanner, 100
 dfvfs.helpers.text_file, 107
 dfvfs.helpers.volume_scanner, 108
 dfvfs.helpers.windows_path_resolver, 112
 dfvfs.lib, 127
 dfvfs.lib.apfs_helper, 113
 dfvfs.lib.bde_helper, 113
 dfvfs.lib.cpio, 114
 dfvfs.lib.cs_helper, 116
 dfvfs.lib.data_format, 117
 dfvfs.lib.decorators, 117
 dfvfs.lib.definitions, 117
 dfvfs.lib.errors, 117
 dfvfs.lib.ewf_helper, 118
 dfvfs.lib.glob2regex, 119
 dfvfs.lib.gpt_helper, 119
 dfvfs.lib.gzipfile, 119
 dfvfs.lib.luksde_helper, 122
 dfvfs.lib.lvm_helper, 123
 dfvfs.lib.raw_helper, 123
 dfvfs.lib.sqlite_database, 123
 dfvfs.lib.tsk_image, 125
 dfvfs.lib.tsk_partition, 126
 dfvfs.lib.vshadow_helper, 127
 dfvfs.mount, 128
 dfvfs.mount.manager, 127
 dfvfs.path, 146
 dfvfs.path.apfs_container_path_spec, 128
 dfvfs.path.apfs_path_spec, 129
 dfvfs.path.bde_path_spec, 129
 dfvfs.path.compressed_stream_path_spec, 130
 dfvfs.path.cpio_path_spec, 130
 dfvfs.path.cs_path_spec, 131
 dfvfs.path.data_range_path_spec, 131
 dfvfs.path.encoded_stream_path_spec, 132
 dfvfs.path.encrypted_stream_path_spec, 132
 dfvfs.path.ewf_path_spec, 133
 dfvfs.path.ext_path_spec, 133
 dfvfs.path.factory, 134
 dfvfs.path.fake_path_spec, 135
 dfvfs.path.fat_path_spec, 135
 dfvfs.path.gpt_path_spec, 136
 dfvfs.path.gzip_path_spec, 136
 dfvfs.path.hfs_path_spec, 137
 dfvfs.path.location_path_spec, 137
 dfvfs.path.luksde_path_spec, 138
 dfvfs.path.lvm_path_spec, 138
 dfvfs.path.modi_path_spec, 139
 dfvfs.path.mount_path_spec, 139
 dfvfs.path.ntfs_path_spec, 139
 dfvfs.path.os_path_spec, 140
 dfvfs.path.path_spec, 140
 dfvfs.path.phdi_path_spec, 142
 dfvfs.path.qcow_path_spec, 142
 dfvfs.path.raw_path_spec, 142
 dfvfs.path.sqlite_blob_path_spec, 142
 dfvfs.path.tar_path_spec, 143
 dfvfs.path.tsk_partition_path_spec, 143
 dfvfs.path.tsk_path_spec, 144
 dfvfs.path.vhdi_path_spec, 145
 dfvfs.path.vmdk_path_spec, 145
 dfvfs.path.vshadow_path_spec, 145
 dfvfs.path.xfs_path_spec, 146
 dfvfs.path.zip_path_spec, 146
 dfvfs.resolver, 149
 dfvfs.resolver.context, 146
 dfvfs.resolver.resolver, 148
 dfvfs.resolver_helpers, 171
 dfvfs.resolver_helpers.apfs_container_resolver_helper, 149
 dfvfs.resolver_helpers.apfs_resolver_helper, 150
 dfvfs.resolver_helpers.bde_resolver_helper, 151
 dfvfs.resolver_helpers.compressed_stream_resolver_helper, 151
 dfvfs.resolver_helpers.cpio_resolver_helper, 152
 dfvfs.resolver_helpers.cs_resolver_helper, 153
 dfvfs.resolver_helpers.data_range_resolver_helper, 153
 dfvfs.resolver_helpers.encoded_stream_resolver_helper, 154
 dfvfs.resolver_helpers.encrypted_stream_resolver_helper, 155
 dfvfs.resolver_helpers.ewf_resolver_helper, 155
 dfvfs.resolver_helpers.ext_resolver_helper, 156
 dfvfs.resolver_helpers.fake_resolver_helper, 156

dfvfs.resolver_helpers.fat_resolver_helper, 157
 dfvfs.resolver_helpers.gpt_resolver_helper, 157
 dfvfs.resolver_helpers.gzip_resolver_helper, 158
 dfvfs.resolver_helpers.hfs_resolver_helper, 159
 dfvfs.resolver_helpers.luksde_resolver_helper, 159
 dfvfs.resolver_helpers.lvm_resolver_helper, 160
 dfvfs.resolver_helpers.manager, 161
 dfvfs.resolver_helpers.modi_resolver_helper, 161
 dfvfs.resolver_helpers.ntfs_resolver_helper, 162
 dfvfs.resolver_helpers.os_resolver_helper, 162
 dfvfs.resolver_helpers.phdi_resolver_helper, 163
 dfvfs.resolver_helpers.qcow_resolver_helper, 164
 dfvfs.resolver_helpers.raw_resolver_helper, 164
 dfvfs.resolver_helpers.resolver_helper, 165
 dfvfs.resolver_helpers.sqlite_blob_resolver_helper, 165
 dfvfs.resolver_helpers.tar_resolver_helper, 166
 dfvfs.resolver_helpers.tsk_partition_resolver_helper, 167
 dfvfs.resolver_helpers.tsk_resolver_helper, 167
 dfvfs.resolver_helpers.vhdi_resolver_helper, 168
 dfvfs.resolver_helpers.vmdk_resolver_helper, 169
 dfvfs.resolver_helpers.vshadow_resolver_helper, 169
 dfvfs.resolver_helpers.xfs_resolver_helper, 170
 dfvfs.resolver_helpers.zip_resolver_helper, 170
 dfvfs.serializer, 172
 dfvfs.serializer.json_serializer, 171
 dfvfs.serializer.serializer, 172
 dfvfs.vfs, 253
 dfvfs.vfs.apfs_attribute, 172
 dfvfs.vfs.apfs_container_directory, 174
 dfvfs.vfs.apfs_container_file_entry, 174
 dfvfs.vfs.apfs_container_file_system, 175
 dfvfs.vfs.apfs_directory, 176
 dfvfs.vfs.apfs_file_entry, 176
 dfvfs.vfs.apfs_file_system, 178
 dfvfs.vfs.attribute, 179
 dfvfs.vfs.bde_file_entry, 181
 dfvfs.vfs.bde_file_system, 181
 dfvfs.vfs.compressed_stream_file_entry, 182
 dfvfs.vfs.compressed_stream_file_system, 182
 dfvfs.vfs.cpio_directory, 183
 dfvfs.vfs.cpio_file_entry, 183
 dfvfs.vfs.cpio_file_system, 184
 dfvfs.vfs.cs_directory, 185
 dfvfs.vfs.cs_file_entry, 185
 dfvfs.vfs.cs_file_system, 186
 dfvfs.vfs.data_range_file_entry, 187
 dfvfs.vfs.data_range_file_system, 188
 dfvfs.vfs.data_stream, 188
 dfvfs.vfs.directory, 189
 dfvfs.vfs.encoded_stream_file_entry, 189
 dfvfs.vfs.encoded_stream_file_system, 189
 dfvfs.vfs.encrypted_stream_file_entry, 190
 dfvfs.vfs.encrypted_stream_file_system, 190
 dfvfs.vfs.ext_attribute, 191
 dfvfs.vfs.ext_directory, 192
 dfvfs.vfs.ext_file_entry, 193
 dfvfs.vfs.ext_file_system, 194
 dfvfs.vfs.extent, 195
 dfvfs.vfs.fake_directory, 196
 dfvfs.vfs.fake_file_entry, 196
 dfvfs.vfs.fake_file_system, 197
 dfvfs.vfs.fat_directory, 199
 dfvfs.vfs.fat_file_entry, 199
 dfvfs.vfs.fat_file_system, 200
 dfvfs.vfs.file_entry, 202
 dfvfs.vfs.file_system, 207
 dfvfs.vfs.gpt_directory, 210
 dfvfs.vfs.gpt_file_entry, 210
 dfvfs.vfs.gpt_file_system, 211
 dfvfs.vfs.gzip_file_entry, 212
 dfvfs.vfs.gzip_file_system, 212
 dfvfs.vfs.hfs_attribute, 213
 dfvfs.vfs.hfs_data_stream, 214
 dfvfs.vfs.hfs_directory, 215
 dfvfs.vfs.hfs_file_entry, 215
 dfvfs.vfs.hfs_file_system, 217
 dfvfs.vfs.luksde_file_entry, 218
 dfvfs.vfs.luksde_file_system, 218
 dfvfs.vfs.lvm_directory, 219
 dfvfs.vfs.lvm_file_entry, 219
 dfvfs.vfs.lvm_file_system, 220
 dfvfs.vfs.ntfs_attribute, 221

dfvfs.vfs.ntfs_data_stream, 223
 dfvfs.vfs.ntfs_directory, 224
 dfvfs.vfs.ntfs_file_entry, 224
 dfvfs.vfs.ntfs_file_system, 226
 dfvfs.vfs.os_attribute, 227
 dfvfs.vfs.os_directory, 228
 dfvfs.vfs.os_file_entry, 229
 dfvfs.vfs.os_file_system, 230
 dfvfs.vfs.root_only_file_entry, 231
 dfvfs.vfs.root_only_file_system, 231
 dfvfs.vfs.sqlite_blob_directory, 232
 dfvfs.vfs.sqlite_blob_file_entry, 232
 dfvfs.vfs.sqlite_blob_file_system, 233
 dfvfs.vfs.tar_directory, 233
 dfvfs.vfs.tar_file_entry, 234
 dfvfs.vfs.tar_file_system, 235
 dfvfs.vfs.tsk_attribute, 236
 dfvfs.vfs.tsk_data_stream, 237
 dfvfs.vfs.tsk_directory, 238
 dfvfs.vfs.tsk_file_entry, 238
 dfvfs.vfs.tsk_file_system, 241
 dfvfs.vfs.tsk_partition_directory, 243
 dfvfs.vfs.tsk_partition_file_entry, 243
 dfvfs.vfs.tsk_partition_file_system, 244
 dfvfs.vfs.vshadow_directory, 245
 dfvfs.vfs.vshadow_file_entry, 245
 dfvfs.vfs.vshadow_file_system, 246
 dfvfs.vfs.xfs_attribute, 247
 dfvfs.vfs.xfs_directory, 248
 dfvfs.vfs.xfs_file_entry, 248
 dfvfs.vfs.xfs_file_system, 250
 dfvfs.vfs.zip_directory, 251
 dfvfs.vfs.zip_file_entry, 251
 dfvfs.vfs.zip_file_system, 252
 dfvfs.volume, 259
 dfvfs.volume.apfs_volume_system, 253
 dfvfs.volume.cs_volume_system, 254
 dfvfs.volume.factory, 254
 dfvfs.volume.gpt_volume_system, 255
 dfvfs.volume.lvm_volume_system, 255
 dfvfs.volume.tsk_volume_system, 255
 dfvfs.volume.volume_system, 256
 dfvfs.volume.vshadow_volume_system, 258
 MountPathSpec (class in *dfvfs.path.mount_path_spec*), 139
 MountPointError, 117
 MountPointManager (class in *dfvfs.mount.manager*), 127
N
 name (*dfvfs.vfs.apfs_attribute.APFSExtendedAttribute property*), 173
 name (*dfvfs.vfs.apfs_container_file_entry.APFSContainerFileEntry method*), 150
 name (*dfvfs.vfs.apfs_file_entry.APFSFileEntry property*), 178
 name (*dfvfs.vfs.cpio_file_entry.CPIOFileEntry property*), 184
 name (*dfvfs.vfs.cs_file_entry.CSFileEntry property*), 186
 name (*dfvfs.vfs.data_stream.DataStream property*), 188
 name (*dfvfs.vfs.ext_attribute.EXTExtendedAttribute property*), 191
 name (*dfvfs.vfs.ext_file_entry.EXTFileEntry property*), 194
 name (*dfvfs.vfs.fake_file_entry.FakeFileEntry property*), 197
 name (*dfvfs.vfs.fat_file_entry.FATFileEntry property*), 200
 name (*dfvfs.vfs.file_entry.FileEntry property*), 206
 name (*dfvfs.vfs.gpt_file_entry.GPTFileEntry property*), 210
 name (*dfvfs.vfs.hfs_attribute.HFSExtendedAttribute property*), 213
 name (*dfvfs.vfs.hfs_file_entry.HFSFileEntry property*), 216
 name (*dfvfs.vfs.lvm_file_entry.LVMFileEntry property*), 219
 name (*dfvfs.vfs.ntfs_attribute.FileNameNTFSAttribute property*), 221
 name (*dfvfs.vfs.ntfs_file_entry.NTFSFileEntry property*), 225
 name (*dfvfs.vfs.os_attribute.OSExtendedAttribute property*), 227
 name (*dfvfs.vfs.os_file_entry.OSFileEntry property*), 229
 name (*dfvfs.vfs.root_only_file_entry.RootOnlyFileEntry property*), 231
 name (*dfvfs.vfs.sqlite_blob_file_entry.SQLiteBlobFileEntry property*), 232
 name (*dfvfs.vfs.tar_file_entry.TARFileEntry property*), 234
 name (*dfvfs.vfs.tsk_attribute.TSKExtendedAttribute property*), 236
 name (*dfvfs.vfs.tsk_file_entry.TSKFileEntry property*), 240
 name (*dfvfs.vfs.tsk_partition_file_entry.TSKPartitionFileEntry property*), 243
 name (*dfvfs.vfs.vshadow_file_entry.VShadowFileEntry property*), 245
 name (*dfvfs.vfs.xfs_attribute.XFSExtendedAttribute property*), 247
 name (*dfvfs.vfs.xfs_file_entry.XFSFileEntry property*), 249
 name (*dfvfs.vfs.zip_file_entry.ZipFileEntry property*), 252
 name_space (*dfvfs.vfs.ntfs_attribute.FileNameNTFSAttribute property*), 222
 NewFileObject () (*dfvfs.resolver_helpers.apfs_resolver_helper.APFSResolverHelper*), 150
 NewFileObject () (*dfvfs.resolver_helpers.bde_resolver_helper.BDEResolverHelper*), 150

method), 151
 NewFileObject() (dfvfs.resolver_helpers.compressed_stream_resolver_helper.VShadowResolverHelper method), 151
 NewFileObject() (dfvfs.resolver_helpers.cpio_resolver_helper.CPIOResolverHelper method), 152
 NewFileObject() (dfvfs.resolver_helpers.cs_resolver_helper.CSResolverHelper method), 153
 NewFileObject() (dfvfs.resolver_helpers.data_range_resolver_helper.DataRangeResolverHelper method), 153
 NewFileObject() (dfvfs.resolver_helpers.encoded_stream_resolver_helper.EncodedStreamResolverHelper method), 154
 NewFileObject() (dfvfs.resolver_helpers.encrypted_stream_resolver_helper.EncryptedStreamResolverHelper method), 155
 NewFileObject() (dfvfs.resolver_helpers.ewf_resolver_helper.EWFResolverHelper method), 155
 NewFileObject() (dfvfs.resolver_helpers.ext_resolver_helper.EXTResolverHelper method), 156
 NewFileObject() (dfvfs.resolver_helpers.fat_resolver_helper.FATResolverHelper method), 157
 NewFileObject() (dfvfs.resolver_helpers.gpt_resolver_helper.GPTRResolverHelper method), 157
 NewFileObject() (dfvfs.resolver_helpers.gzip_resolver_helper.GzipResolverHelper method), 158
 NewFileObject() (dfvfs.resolver_helpers.hfs_resolver_helper.HFSResolverHelper method), 159
 NewFileObject() (dfvfs.resolver_helpers.luksde_resolver_helper.LUKSDResolverHelper method), 159
 NewFileObject() (dfvfs.resolver_helpers.lvm_resolver_helper.LVMResolverHelper method), 160
 NewFileObject() (dfvfs.resolver_helpers.modi_resolver_helper.MODIResolverHelper method), 161
 NewFileObject() (dfvfs.resolver_helpers.ntfs_resolver_helper.NTFSResolverHelper method), 162
 NewFileObject() (dfvfs.resolver_helpers.os_resolver_helper.OSResolverHelper method), 162
 NewFileObject() (dfvfs.resolver_helpers.phdi_resolver_helper.PHDIResolverHelper method), 163
 NewFileObject() (dfvfs.resolver_helpers.qcow_resolver_helper.QCOWResolverHelper method), 164
 NewFileObject() (dfvfs.resolver_helpers.raw_resolver_helper.RawResolverHelper method), 164
 NewFileObject() (dfvfs.resolver_helpers.resolver_helper.ResolverHelper method), 165
 NewFileObject() (dfvfs.resolver_helpers.sqlite_blob_resolver_helper.SQLiteBlobResolverHelper method), 165
 NewFileObject() (dfvfs.resolver_helpers.tar_resolver_helper.TARResolverHelper method), 166
 NewFileObject() (dfvfs.resolver_helpers.tsk_partition_resolver_helper.TSKPartitionResolverHelper method), 167
 NewFileObject() (dfvfs.resolver_helpers.tsk_resolver_helper.TSKResolverHelper method), 167
 NewFileObject() (dfvfs.resolver_helpers.vhdi_resolver_helper.VHDIResolverHelper method), 168
 NewFileObject() (dfvfs.resolver_helpers.vmdk_resolver_helper.VMDKResolverHelper method), 168
 NewFileObject() (dfvfs.resolver_helpers.vshadow_resolver_helper.VShadowResolverHelper method), 169
 NewFileObject() (dfvfs.resolver_helpers.xfs_resolver_helper.XFSResolverHelper method), 170
 NewFileObject() (dfvfs.resolver_helpers.zip_resolver_helper.ZipResolverHelper method), 170
 NewFileObject() (dfvfs.resolver_helpers.apfs_container_resolver_helper.APFSContainerResolverHelper method), 149
 NewFileObject() (dfvfs.resolver_helpers.apfs_resolver_helper.APFSResolverHelper method), 150
 NewFileObject() (dfvfs.resolver_helpers.bder_resolver_helper.BDERResolverHelper method), 151
 NewFileObject() (dfvfs.resolver_helpers.compressed_stream_resolver_helper.CPIOResolverHelper method), 152
 NewFileObject() (dfvfs.resolver_helpers.cpio_resolver_helper.CPIOResolverHelper method), 152
 NewFileObject() (dfvfs.resolver_helpers.cs_resolver_helper.CSResolverHelper method), 153
 NewFileObject() (dfvfs.resolver_helpers.data_range_resolver_helper.DataRangeResolverHelper method), 154
 NewFileObject() (dfvfs.resolver_helpers.encoded_stream_resolver_helper.EncodedStreamResolverHelper method), 154
 NewFileObject() (dfvfs.resolver_helpers.encrypted_stream_resolver_helper.EncryptedStreamResolverHelper method), 155
 NewFileObject() (dfvfs.resolver_helpers.ext_resolver_helper.EXTResolverHelper method), 156
 NewFileObject() (dfvfs.resolver_helpers.fat_resolver_helper.FATResolverHelper method), 157
 NewFileObject() (dfvfs.resolver_helpers.gpt_resolver_helper.GPTRResolverHelper method), 158
 NewFileObject() (dfvfs.resolver_helpers.gzip_resolver_helper.GzipResolverHelper method), 158
 NewFileObject() (dfvfs.resolver_helpers.hfs_resolver_helper.HFSResolverHelper method), 159
 NewFileObject() (dfvfs.resolver_helpers.luksde_resolver_helper.LUKSDResolverHelper method), 160
 NewFileObject() (dfvfs.resolver_helpers.lvm_resolver_helper.LVMResolverHelper method), 160
 NewFileObject() (dfvfs.resolver_helpers.modi_resolver_helper.MODIResolverHelper method), 161
 NewFileObject() (dfvfs.resolver_helpers.ntfs_resolver_helper.NTFSResolverHelper method), 162
 NewFileObject() (dfvfs.resolver_helpers.os_resolver_helper.OSResolverHelper method), 163
 NewFileObject() (dfvfs.resolver_helpers.phdi_resolver_helper.PHDIResolverHelper method), 163
 NewFileObject() (dfvfs.resolver_helpers.qcow_resolver_helper.QCOWResolverHelper method), 164
 NewFileObject() (dfvfs.resolver_helpers.raw_resolver_helper.RawResolverHelper method), 164
 NewFileObject() (dfvfs.resolver_helpers.resolver_helper.ResolverHelper method), 165
 NewFileObject() (dfvfs.resolver_helpers.sqlite_blob_resolver_helper.SQLiteBlobResolverHelper method), 165
 NewFileObject() (dfvfs.resolver_helpers.tar_resolver_helper.TARResolverHelper method), 166
 NewFileObject() (dfvfs.resolver_helpers.tsk_partition_resolver_helper.TSKPartitionResolverHelper method), 166
 NewFileObject() (dfvfs.resolver_helpers.tsk_resolver_helper.TSKResolverHelper method), 167
 NewFileObject() (dfvfs.resolver_helpers.vhdi_resolver_helper.VHDIResolverHelper method), 168
 NewFileObject() (dfvfs.resolver_helpers.vmdk_resolver_helper.VMDKResolverHelper method), 168
 NewFileObject() (dfvfs.resolver_helpers.vshadow_resolver_helper.VShadowResolverHelper method), 169

- method), 169
- NewFileSystem() (dfvfs.resolver_helpers.xfs_resolver_helper.XFSResolverHelper method), 170
- NewFileSystem() (dfvfs.resolver_helpers.zip_resolver_helper.ZipResolverHelper method), 171
- NewPathSpec() (dfvfs.path.factory.Factory class method), 134
- NewVolumeSystem() (dfvfs.volume.factory.Factory class method), 254
- NotSupported, 118
- NTFSAnalyzerHelper (class in dfvfs.analyzer.ntfs_analyzer_helper), 38
- NTFSAttribute (class in dfvfs.vfs.ntfs_attribute), 222
- NTFSDataStream (class in dfvfs.vfs.ntfs_data_stream), 223
- NTFSDirectory (class in dfvfs.vfs.ntfs_directory), 224
- NTFSFile (class in dfvfs.file_io.ntfs_file_io), 80
- NTFSFileEntry (class in dfvfs.vfs.ntfs_file_entry), 224
- NTFSFileSystem (class in dfvfs.vfs.ntfs_file_system), 226
- NTFSPathSpec (class in dfvfs.path.ntfs_path_spec), 139
- NTFSResolverHelper (class in dfvfs.resolver_helpers.ntfs_resolver_helper), 162
- number_of_attributes (dfvfs.vfs.file_entry.FileEntry property), 206
- number_of_attributes (dfvfs.volume.volume_system.Volume property), 256
- number_of_data_streams (dfvfs.vfs.file_entry.FileEntry property), 207
- number_of_extents (dfvfs.volume.volume_system.Volume property), 256
- number_of_links (dfvfs.lib.cpio.CPIOArchiveFileEntry attribute), 116
- number_of_links (dfvfs.vfs.attribute.StatAttribute attribute), 180
- number_of_sections (dfvfs.volume.volume_system.VolumeSystem property), 258
- number_of_sub_file_entries (dfvfs.vfs.file_entry.FileEntry property), 207
- number_of_volumes (dfvfs.volume.volume_system.VolumeSystem property), 258
- O**
- ObjectIdentifierNTFSAttribute (class in dfvfs.vfs.ntfs_attribute), 222
- offset (dfvfs.analyzer.specification.Signature attribute), 41
- offset (dfvfs.vfs.extent.Extent attribute), 195
- Open() (dfvfs.file_io.file_io.FileIO method), 72
- Open() (dfvfs.lib.cpio.CPIOArchiveFile method), 114
- Open() (dfvfs.lib.gzipfile.GzipCompressedStream method), 119
- Open() (dfvfs.lib.sqlite_database.SQLiteDatabaseFile method), 124
- Open() (dfvfs.vfs.file_system.FileSystem method), 209
- Open() (dfvfs.volume.volume_system.VolumeSystem method), 257
- OpenFile() (dfvfs.helpers.volume_scanner.WindowsVolumeScanner method), 111
- OpenFileEntry() (dfvfs.resolver.resolver.Resolver class method), 148
- OpenFileObject() (dfvfs.resolver.resolver.Resolver class method), 148
- OpenFileSystem() (dfvfs.resolver.resolver.Resolver class method), 149
- OpenSourcePath() (dfvfs.helpers.source_scanner.SourceScannerContext method), 106
- operating_system (dfvfs.lib.gzipfile.GzipMember attribute), 121
- operating_systems (dfvfs.file_io.gzip_file_io.GzipFile property), 76
- original_filename (dfvfs.lib.gzipfile.GzipMember attribute), 121
- original_filenames (dfvfs.file_io.gzip_file_io.GzipFile property), 76
- OSDirectory (class in dfvfs.vfs.os_directory), 228
- OSExtendedAttribute (class in dfvfs.vfs.os_attribute), 227
- OSFile (class in dfvfs.file_io.os_file_io), 81
- OSFileEntry (class in dfvfs.vfs.os_file_entry), 229
- OSFileSystem (class in dfvfs.vfs.os_file_system), 230
- OSPathSpec (class in dfvfs.path.os_path_spec), 140
- OSResolverHelper (class in dfvfs.resolver_helpers.os_resolver_helper), 162
- owner_identifier (dfvfs.vfs.attribute.StatAttribute attribute), 180
- owner_identifier (dfvfs.vfs.ntfs_attribute.StandardInformationNTFSAttribute attribute), 223
- P**
- parent (dfvfs.path.path_spec.PathSpec attribute), 140
- parent_file_reference (dfvfs.vfs.ntfs_attribute.FileNameNTFSAttribute property), 222
- parent_node (dfvfs.helpers.source_scanner.SourceScanNode attribute), 100
- ParseVolumeIdentifiersString() (dfvfs.helpers.command_line.CLIVolumeScannerMediator method), 94
- part_index (dfvfs.path.tsk_partition_path_spec.TSKPartitionPathSpec attribute), 143
- partitions (dfvfs.helpers.volume_scanner.VolumeScannerOptions attribute), 110
- password (dfvfs.path.bde_path_spec.BDEPathSpec attribute), 129

password (*dfvfs.path.cs_path_spec.CSPathSpec* attribute), 131

password (*dfvfs.path.luksde_path_spec.LUKSDEPathSpec* attribute), 138

path (*dfvfs.lib.cpio.CPIOArchiveFileEntry* attribute), 116

PATH_SEPARATOR (*dfvfs.vfs.fat_file_system.FATFilesystem* attribute), 201

PATH_SEPARATOR (*dfvfs.vfs.file_system.FileSystem* attribute), 209

PATH_SEPARATOR (*dfvfs.vfs.ntfs_file_system.NTFSFilesystem* attribute), 227

PATH_SEPARATOR (*dfvfs.vfs.os_file_system.OSFilesystem* attribute), 230

path_spec (*dfvfs.helpers.source_scanner.SourceScanNode* attribute), 100

path_spec (*dfvfs.vfs.directory.Directory* attribute), 189

path_spec (*dfvfs.vfs.file_entry.FileEntry* attribute), 202

PathSpec (class in *dfvfs.path.path_spec*), 140

PathSpecError, 118

PathSpecSerializer (class in *dfvfs.serializer.serializer*), 172

pattern (*dfvfs.analyzer.specification.Signature* attribute), 41

PHDIAnalyzerHelper (class in *dfvfs.analyzer.phdi_analyzer_helper*), 39

PHDIFile (class in *dfvfs.file_io.phdi_file_io*), 82

PHDIPathSpec (class in *dfvfs.path.phdi_path_spec*), 142

PHDIResolverHelper (class in *dfvfs.resolver_helpers.phdi_resolver_helper*), 163

PrintWarning() (*dfvfs.helpers.command_line.CLIVolumeScannerMethod*), 94

PROPERTY_NAMES (*dfvfs.path.factory.Factory* attribute), 135

Q

QCOWAnalyzerHelper (class in *dfvfs.analyzer.qcow_analyzer_helper*), 39

QCOWFile (class in *dfvfs.file_io.qcow_file_io*), 82

QCOWPathSpec (class in *dfvfs.path.qcow_path_spec*), 142

QCOWResolverHelper (class in *dfvfs.resolver_helpers.qcow_resolver_helper*), 164

Query() (*dfvfs.lib.sqlite_database.SQLiteDatabaseFile* method), 125

R

range_offset (*dfvfs.path.data_range_path_spec.DataRangePathSpec* attribute), 131

range_size (*dfvfs.path.data_range_path_spec.DataRangePathSpec* attribute), 132

RawFile (class in *dfvfs.file_io.raw_file_io*), 83

RawGlobPathSpec() (in module *dfvfs.lib.raw_helper*), 123

RawPathSpec (class in *dfvfs.path.raw_path_spec*), 142

RawResolverHelper (class in *dfvfs.resolver_helpers.raw_resolver_helper*), 164

RC4Decrypter (class in *dfvfs.encryption.rc4_decrypter*), 58

read() (*dfvfs.file_io.apfs_file_io.APFSFile* method), 59

read() (*dfvfs.file_io.compressed_stream_io.CompressedStream* method), 60

read() (*dfvfs.file_io.cpio_file_io.CPIOFile* method), 62

read() (*dfvfs.file_io.cs_file_io.CSFile* method), 63

read() (*dfvfs.file_io.data_range_io.DataRange* method), 64

read() (*dfvfs.file_io.encoded_stream_io.EncodedStream* method), 66

read() (*dfvfs.file_io.encrypted_stream_io.EncryptedStream* method), 67

read() (*dfvfs.file_io.ext_file_io.EXTFile* method), 68

read() (*dfvfs.file_io.fake_file_io.FakeFile* method), 70

read() (*dfvfs.file_io.fat_file_io.FATFile* method), 71

read() (*dfvfs.file_io.file_io.FileIO* method), 72

read() (*dfvfs.file_io.file_object_io.FileObjectIO* method), 74

read() (*dfvfs.file_io.gpt_file_io.GPTFile* method), 75

read() (*dfvfs.file_io.hfs_file_io.HFSFile* method), 77

read() (*dfvfs.file_io.lvm_file_io.LVMFile* method), 78

read() (*dfvfs.file_io.ntfs_file_io.NTFSFile* method), 80

read() (*dfvfs.file_io.os_file_io.OSFile* method), 81

read() (*dfvfs.file_io.sqlite_blob_file_io.SQLiteBlobFile* method), 84

read() (*dfvfs.file_io.tar_file_io.TARFile* method), 85

read() (*dfvfs.file_io.tsk_file_io.TSKFile* method), 86

read() (*dfvfs.file_io.vshadow_file_io.VShadowFile* method), 88

read() (*dfvfs.file_io.xfs_file_io.XFSFile* method), 90

read() (*dfvfs.file_io.zip_file_io.ZipFile* method), 91

Read() (*dfvfs.helpers.command_line.CLIInputReader* method), 92

Read() (*dfvfs.helpers.command_line.FileObjectInputReader* method), 94

read() (*dfvfs.lib.zipfile.GzipCompressedStream* method), 120

read() (*dfvfs.lib.tsk_image.TSKFilesystemImage* method), 125

read() (*dfvfs.vfs.apfs_attribute.APFSExtendedAttribute* method), 173

read() (*dfvfs.vfs.ext_attribute.EXTExtendedAttribute* method), 191

read() (*dfvfs.vfs.hfs_attribute.HFSExtendedAttribute* method), 213

read() (*dfvfs.vfs.os_attribute.OSExtendedAttribute* method), 227

read() (dfvfs.vfs.tsk_attribute.TSKExtendedAttribute method), 106
 read() (dfvfs.vfs.xfs_attribute.XFSExtendedAttribute method), 236
 read() (dfvfs.vfs.xfs_attribute.XFSExtendedAttribute method), 247
 ReadAtOffset() (dfvfs.lib.zipfile.GzipMember method), 122
 ReadDataAtOffset() (dfvfs.lib.cpio.CPIOArchiveFile method), 115
 readline() (dfvfs.helpers.text_file.TextFile method), 107
 readlines() (dfvfs.helpers.text_file.TextFile method), 107
 ReadSerialized() (dfvfs.serializer.json_serializer.JsonPathSpecSerializer class method), 171
 ReadSerialized() (dfvfs.serializer.serializer.PathSpecSerializer method), 172
 recovery_password (dfvfs.path.bde_path_spec.BDEPathSpec attribute), 129
 recovery_password (dfvfs.path.cs_path_spec.CSPathSpec attribute), 131
 RegisterCredentials() (dfvfs.credentials.manager.CredentialsManager class method), 52
 RegisterDecoder() (dfvfs.encoding.manager.EncodingManager class method), 55
 RegisterDecoders() (dfvfs.encoding.manager.EncodingManager class method), 55
 RegisterDecompressor() (dfvfs.compression.manager.CompressionManager class method), 47
 RegisterDecompressors() (dfvfs.compression.manager.CompressionManager class method), 47
 RegisterDecrypter() (dfvfs.encryption.manager.EncryptionManager class method), 57
 RegisterDecrypters() (dfvfs.encryption.manager.EncryptionManager class method), 57
 RegisterHelper() (dfvfs.analyzer.analyzer.Analyzer class method), 30
 RegisterHelper() (dfvfs.resolver_helpers.manager.ResolverHelperManager class method), 161
 RegisterMountPoint() (dfvfs.mount.manager.MountPointManager class method), 128
 RegisterMountPoint() (dfvfs.resolver.context.Context method), 148
 RegisterPathSpec() (dfvfs.path.factory.Factory class method), 135
 RegisterVolumeSystem() (dfvfs.volume.factory.Factory class method), 254
 RemoveScanNode() (dfvfs.helpers.source_scanner.SourceScanner class method), 118
 ResolvePath() (dfvfs.helpers.windows_path_resolver.WindowsPathResolver method), 112
 Resolver (class in dfvfs.resolver.resolver), 148
 ResolverHelper (class in dfvfs.resolver_helpers.resolver_helper), 165
 ResolverHelperManager (class in dfvfs.resolver_helpers.manager), 161
 ROOT_DIRECTORY_IDENTIFIER (dfvfs.vfs.apfs_file_system.APFSFileSystem attribute), 179
 ROOT_DIRECTORY_IDENTIFIER_NUMBER (dfvfs.vfs.apfs_file_system.APFSFileSystem attribute), 217
 ROOT_DIRECTORY_INODE_NUMBER (dfvfs.vfs.ext_file_system.EXTFileSystem attribute), 195
 RootOnlyFileEntry (class in dfvfs.vfs.root_only_file_entry), 231
 RootOnlyFileSystem (class in dfvfs.vfs.root_only_file_system), 231
 row_condition (dfvfs.path.sqlite_blob_path_spec.SQLiteBlobPathSpec attribute), 142
 row_index (dfvfs.path.sqlite_blob_path_spec.SQLiteBlobPathSpec attribute), 143
S
 Scan() (dfvfs.helpers.source_scanner.SourceScanner method), 102
 scan_mode (dfvfs.helpers.volume_scanner.VolumeScannerOptions attribute), 110
 SCAN_MODE_ALL (dfvfs.helpers.volume_scanner.VolumeScannerOptions attribute), 111
 SCAN_MODE_SNAPSHOTS_ONLY (dfvfs.helpers.volume_scanner.VolumeScannerOptions attribute), 111
 SCAN_MODE_VOLUMES_ONLY (dfvfs.helpers.volume_scanner.VolumeScannerOptions attribute), 111
 ScanForFileSystem() (dfvfs.helpers.source_scanner.SourceScanner method), 102
 ScanForStorageMediaImage() (dfvfs.helpers.source_scanner.SourceScanner method), 102
 ScanForVolumeSystem() (dfvfs.helpers.source_scanner.SourceScanner method), 103
 ScanForWindowsVolume() (dfvfs.helpers.volume_scanner.WindowsVolumeScanner method), 111
 scanned (dfvfs.helpers.source_scanner.SourceScanNode attribute), 100
 ScannerError (class in dfvfs.helpers.source_scanner), 118

- sections (*dfvfs.volume.volume_system.VolumeSystem* property), 258
- security_descriptor (*dfvfs.vfs.ntfs_attribute.SecurityDescriptorNTFSAttribute* property), 222
- security_descriptor_identifer (*dfvfs.vfs.ntfs_attribute.StandardInformationNTFSAttribute* method), 214
- SecurityDescriptorNTFSAttribute (class in *dfvfs.vfs.ntfs_attribute*), 222
- seek() (*dfvfs.file_io.apfs_file_io.APFSFile* method), 59
- seek() (*dfvfs.file_io.compressed_stream_io.CompressedStreamFile* method), 61
- seek() (*dfvfs.file_io.cpio_file_io.CPIOFile* method), 62
- seek() (*dfvfs.file_io.cs_file_io.CSFile* method), 63
- seek() (*dfvfs.file_io.data_range_io.DataRange* method), 64
- seek() (*dfvfs.file_io.encoded_stream_io.EncodedStream* method), 66
- seek() (*dfvfs.file_io.encrypted_stream_io.EncryptedStream* method), 67
- seek() (*dfvfs.file_io.ext_file_io.EXTFile* method), 69
- seek() (*dfvfs.file_io.fake_file_io.FakeFile* method), 70
- seek() (*dfvfs.file_io.fat_file_io.FATFile* method), 71
- seek() (*dfvfs.file_io.file_io.FileIO* method), 73
- seek() (*dfvfs.file_io.file_object_io.FileObjectIO* method), 74
- seek() (*dfvfs.file_io.gpt_file_io.GPTFile* method), 75
- seek() (*dfvfs.file_io.hfs_file_io.HFSFile* method), 77
- seek() (*dfvfs.file_io.lvm_file_io.LVMFile* method), 79
- seek() (*dfvfs.file_io.ntfs_file_io.NTFSFile* method), 80
- seek() (*dfvfs.file_io.os_file_io.OSFile* method), 82
- seek() (*dfvfs.file_io.sqlite_blob_file_io.SQLiteBlobFile* method), 84
- seek() (*dfvfs.file_io.tar_file_io.TARFile* method), 85
- seek() (*dfvfs.file_io.tsk_file_io.TSKFile* method), 87
- seek() (*dfvfs.file_io.vshadow_file_io.VShadowFile* method), 89
- seek() (*dfvfs.file_io.xfs_file_io.XFSFile* method), 90
- seek() (*dfvfs.file_io.zip_file_io.ZipFile* method), 91
- seek() (*dfvfs.lib.gzipfile.GzipCompressedStream* method), 121
- seek() (*dfvfs.vfs.apfs_attribute.APFSExtendedAttribute* method), 173
- seek() (*dfvfs.vfs.ext_attribute.EXTExtendedAttribute* method), 192
- seek() (*dfvfs.vfs.hfs_attribute.HFSExtendedAttribute* method), 214
- seek() (*dfvfs.vfs.os_attribute.OSExtendedAttribute* method), 228
- seek() (*dfvfs.vfs.tsk_attribute.TSKExtendedAttribute* method), 237
- seek() (*dfvfs.vfs.xfs_attribute.XFSExtendedAttribute* method), 248
- seekable() (*dfvfs.file_io.file_io.FileIO* method), 73
- seekable() (*dfvfs.vfs.apfs_attribute.APFSExtendedAttribute* method), 173
- seekable() (*dfvfs.vfs.ext_attribute.EXTExtendedAttribute* method), 192
- seekable() (*dfvfs.vfs.hfs_attribute.HFSExtendedAttribute* method), 214
- seekable() (*dfvfs.vfs.os_attribute.OSExtendedAttribute* method), 228
- seekable() (*dfvfs.vfs.tsk_attribute.TSKExtendedAttribute* method), 237
- seekable() (*dfvfs.vfs.xfs_attribute.XFSExtendedAttribute* method), 248
- SetCredential() (*dfvfs.credentials.keychain.KeyChain* method), 51
- SetDecodedStreamSize() (*dfvfs.file_io.encoded_stream_io.EncodedStream* method), 65
- SetDecryptedStreamSize() (*dfvfs.file_io.encrypted_stream_io.EncryptedStream* method), 66
- SetEnvironmentVariable() (*dfvfs.helpers.windows_path_resolver.WindowsPathResolver* method), 112
- SetIdentifier() (*dfvfs.analyzer.specification.Signature* method), 41
- SetSourceType() (*dfvfs.helpers.source_scanner.SourceScannerContext* method), 106
- Signature (class in *dfvfs.analyzer.specification*), 41
- size (*dfvfs.lib.cpio.CPIOArchiveFileEntry* attribute), 116
- size (*dfvfs.vfs.apfs_container_file_entry.APFSContainerFileEntry* property), 175
- size (*dfvfs.vfs.apfs_file_entry.APFSFileEntry* property), 178
- size (*dfvfs.vfs.attribute.StatAttribute* attribute), 180
- size (*dfvfs.vfs.bde_file_entry.BDEFileEntry* property), 181
- size (*dfvfs.vfs.compressed_stream_file_entry.CompressedStreamFileEntry* property), 182
- size (*dfvfs.vfs.cpio_file_entry.CPIOFileEntry* property), 184
- size (*dfvfs.vfs.cs_file_entry.CSFileEntry* property), 186
- size (*dfvfs.vfs.data_range_file_entry.DataRangeFileEntry* property), 187
- size (*dfvfs.vfs.encoded_stream_file_entry.EncodedStreamFileEntry* property), 189
- size (*dfvfs.vfs.encrypted_stream_file_entry.EncryptedStreamFileEntry* property), 190
- size (*dfvfs.vfs.ext_file_entry.EXTFileEntry* property), 194
- size (*dfvfs.vfs.extent.Extent* attribute), 195
- size (*dfvfs.vfs.fake_file_entry.FakeFileEntry* property), 197

size (dfvfs.vfs.fat_file_entry.FATFileEntry property), 200
 size (dfvfs.vfs.file_entry.FileEntry property), 207
 size (dfvfs.vfs.gpt_file_entry.GPTFileEntry property), 210
 size (dfvfs.vfs.zip_file_entry.ZipFileEntry property), 212
 size (dfvfs.vfs.hfs_file_entry.HFSFileEntry property), 216
 size (dfvfs.vfs.luksde_file_entry.LUKSDEFileEntry property), 218
 size (dfvfs.vfs.lvm_file_entry.LVMFileEntry property), 220
 size (dfvfs.vfs.ntfs_file_entry.NTFSFileEntry property), 226
 size (dfvfs.vfs.os_file_entry.OSFileEntry property), 229
 size (dfvfs.vfs.sqlite_blob_file_entry.SQLiteBlobFileEntry property), 232
 size (dfvfs.vfs.tar_file_entry.TARFileEntry property), 234
 size (dfvfs.vfs.tsk_file_entry.TSKFileEntry property), 240
 size (dfvfs.vfs.tsk_partition_file_entry.TSKPartitionFileEntry property), 243
 size (dfvfs.vfs.vshadow_file_entry.VShadowFileEntry property), 245
 size (dfvfs.vfs.xfs_file_entry.XFSFileEntry property), 250
 size (dfvfs.vfs.zip_file_entry.ZipFileEntry property), 252
 snapshots (dfvfs.helpers.volume_scanner.VolumeScannerOptions attribute), 111
 source_type (dfvfs.helpers.source_scanner.SourceScannerContext property), 103
 SOURCE_TYPE_DIRECTORY (dfvfs.helpers.source_scanner.SourceScannerContext attribute), 106
 SOURCE_TYPE_FILE (dfvfs.helpers.source_scanner.SourceScannerContext attribute), 106
 SOURCE_TYPE_STORAGE_MEDIA_DEVICE (dfvfs.helpers.source_scanner.SourceScannerContext attribute), 106
 SOURCE_TYPE_STORAGE_MEDIA_IMAGE (dfvfs.helpers.source_scanner.SourceScannerContext attribute), 106
 SourceScanner (class in dfvfs.helpers.source_scanner), 101
 SourceScannerContext (class in dfvfs.helpers.source_scanner), 103
 SourceScanNode (class in dfvfs.helpers.source_scanner), 100
 specifications (dfvfs.analyzer.specification.FormatSpecification property), 41
 SplitPath() (dfvfs.helpers.file_system_searcher.FileSystemSearcher method), 98
 SplitPath() (dfvfs.vfs.file_system.FileSystem method), 209
 SQLiteBlobDirectory (class in dfvfs.vfs.sqlite_blob_directory), 232
 SQLiteBlobFile (class in dfvfs.file_io.sqlite_blob_file_io), 83
 SQLiteBlobFileEntry (class in dfvfs.vfs.sqlite_blob_file_entry), 232
 SQLiteBlobFileSystem (class in dfvfs.vfs.sqlite_blob_file_system), 233
 SQLiteBlobPathSpec (class in dfvfs.path.sqlite_blob_path_spec), 142
 SQLiteBlobResolverHelper (class in dfvfs.resolver_helpers.sqlite_blob_resolver_helper), 165
 SQLiteDatabaseFile (class in dfvfs.lib.sqlite_database), 123
 StandardInformationNTFSAttribute (class in dfvfs.vfs.ntfs_attribute), 222
 start_offset (dfvfs.path.tsk_partition_path_spec.TSKPartitionPathSpec attribute), 144
 startup_key (dfvfs.path.bde_path_spec.BDEPathSpec attribute), 129
 StatAttribute (class in dfvfs.vfs.attribute), 179
 StdinInputReader (class in dfvfs.helpers.command_line), 95
 StdoutOutputWriter (class in dfvfs.helpers.command_line), 95
 store_index (dfvfs.path.vshadow_path_spec.VShadowPathSpec attribute), 145
 sub_file_entries (dfvfs.vfs.apfs_container_file_entry.APFSContainerFileEntry attribute), 111
 sub_file_entries (dfvfs.vfs.file_entry.FileEntry property), 207
 sub_nodes (dfvfs.helpers.source_scanner.SourceScanNode attribute), 100
 SupportSizeEncryption() (dfvfs.helpers.source_scanner.SourceScanNode method), 101
T
 table_name (dfvfs.path.sqlite_blob_path_spec.SQLiteBlobPathSpec attribute), 143
 TARAnalyzerHelper (class in dfvfs.analyzer.tar_analyzer_helper), 42
 TARDirectory (class in dfvfs.vfs.tar_directory), 233
 TARFile (class in dfvfs.file_io.tar_file_io), 85
 TARFileEntry (class in dfvfs.vfs.tar_file_entry), 234
 TARFileSystem (class in dfvfs.vfs.tar_file_system), 235
 TARPathSpec (class in dfvfs.path.tar_path_spec), 143
 TARResolverHelper (class in dfvfs.resolver_helpers.tar_resolver_helper), 166
 tell() (dfvfs.file_io.file_io.FileIO method), 73

- `tell()` (*dfvfs.helpers.text_file.TextFile* method), 108
- `tell()` (*dfvfs.vfs.apfs_attribute.APFSExtendedAttribute* method), 174
- `tell()` (*dfvfs.vfs.ext_attribute.EXTExtendedAttribute* method), 192
- `tell()` (*dfvfs.vfs.hfs_attribute.HFSExtendedAttribute* method), 214
- `tell()` (*dfvfs.vfs.os_attribute.OSExtendedAttribute* method), 228
- `tell()` (*dfvfs.vfs.tsk_attribute.TSKExtendedAttribute* method), 237
- `tell()` (*dfvfs.vfs.xfs_attribute.XFSExtendedAttribute* method), 248
- `TextFile` (class in *dfvfs.helpers.text_file*), 107
- `timestamp` (*dfvfs.vfs.tsk_file_entry.TSKTime* property), 241
- `TSKAnalyzerHelper` (class in *dfvfs.analyzer.tsk_analyzer_helper*), 42
- `TSKAttribute` (class in *dfvfs.vfs.tsk_attribute*), 236
- `TSKDataStream` (class in *dfvfs.vfs.tsk_data_stream*), 237
- `TSKDirectory` (class in *dfvfs.vfs.tsk_directory*), 238
- `TSKExtendedAttribute` (class in *dfvfs.vfs.tsk_attribute*), 236
- `TSKFile` (class in *dfvfs.file_io.tsk_file_io*), 86
- `TSKFileEntry` (class in *dfvfs.vfs.tsk_file_entry*), 238
- `TSKFileSystem` (class in *dfvfs.vfs.tsk_file_system*), 241
- `TSKFileSystemImage` (class in *dfvfs.lib.tsk_image*), 125
- `TSKPartitionAnalyzerHelper` (class in *dfvfs.analyzer.tsk_partition_analyzer_helper*), 42
- `TSKPartitionDirectory` (class in *dfvfs.vfs.tsk_partition_directory*), 243
- `TSKPartitionFile` (class in *dfvfs.file_io.tsk_partition_file_io*), 87
- `TSKPartitionFileEntry` (class in *dfvfs.vfs.tsk_partition_file_entry*), 243
- `TSKPartitionFileSystem` (class in *dfvfs.vfs.tsk_partition_file_system*), 244
- `TSKPartitionPathSpec` (class in *dfvfs.path.tsk_partition_path_spec*), 143
- `TSKPartitionResolverHelper` (class in *dfvfs.resolver_helpers.tsk_partition_resolver_helper*), 167
- `TSKPathSpec` (class in *dfvfs.path.tsk_path_spec*), 144
- `TSKResolverHelper` (class in *dfvfs.resolver_helpers.tsk_resolver_helper*), 167
- `TSKTime` (class in *dfvfs.vfs.tsk_file_entry*), 240
- `TSKVolume` (class in *dfvfs.volume.tsk_volume_system*), 255
- `TSKVolumeGetBytesPerSector()` (in module *dfvfs.lib.tsk_partition*), 126
- `TSKVolumeSystem` (class in *dfvfs.volume.tsk_volume_system*), 255
- `TSKVsPartGetNumberOfSectors()` (in module *dfvfs.lib.tsk_partition*), 126
- `TSKVsPartGetStartSector()` (in module *dfvfs.lib.tsk_partition*), 126
- `TSKVsPartIsAllocated()` (in module *dfvfs.lib.tsk_partition*), 126
- `type` (*dfvfs.vfs.attribute.StatAttribute* attribute), 180
- `TYPE_BLOCK_DEVICE` (*dfvfs.vfs.attribute.StatAttribute* attribute), 180
- `TYPE_CHARACTER_DEVICE` (*dfvfs.vfs.attribute.StatAttribute* attribute), 180
- `TYPE_DEVICE` (*dfvfs.vfs.attribute.StatAttribute* attribute), 180
- `TYPE_DIRECTORY` (*dfvfs.vfs.attribute.StatAttribute* attribute), 180
- `TYPE_FILE` (*dfvfs.vfs.attribute.StatAttribute* attribute), 180
- `type_indicator` (*dfvfs.analyzer.analyzer_helper.AnalyzerHelper* property), 31
- `TYPE_INDICATOR` (*dfvfs.analyzer.apfs_analyzer_helper.APFSAAnalyzerHelper* attribute), 32
- `TYPE_INDICATOR` (*dfvfs.analyzer.apfs_container_analyzer_helper.APFSCOAnalyzerHelper* attribute), 32
- `TYPE_INDICATOR` (*dfvfs.analyzer.bde_analyzer_helper.BDEAnalyzerHelper* attribute), 33
- `TYPE_INDICATOR` (*dfvfs.analyzer.bzip2_analyzer_helper.BZIP2AnalyzerHelper* attribute), 33
- `TYPE_INDICATOR` (*dfvfs.analyzer.cpio_analyzer_helper.CPIOAnalyzerHelper* attribute), 33
- `TYPE_INDICATOR` (*dfvfs.analyzer.cs_analyzer_helper.CSAnalyzerHelper* attribute), 34
- `TYPE_INDICATOR` (*dfvfs.analyzer.ewf_analyzer_helper.EWFAAnalyzerHelper* attribute), 34
- `TYPE_INDICATOR` (*dfvfs.analyzer.ext_analyzer_helper.EXTAnalyzerHelper* attribute), 35
- `TYPE_INDICATOR` (*dfvfs.analyzer.fat_analyzer_helper.FATANalyzerHelper* attribute), 35
- `TYPE_INDICATOR` (*dfvfs.analyzer.gpt_analyzer_helper.GPTAnalyzerHelper* attribute), 36
- `TYPE_INDICATOR` (*dfvfs.analyzer.gzip_analyzer_helper.GzipAnalyzerHelper* attribute), 36
- `TYPE_INDICATOR` (*dfvfs.analyzer.hfs_analyzer_helper.HFSAAnalyzerHelper* attribute), 37
- `TYPE_INDICATOR` (*dfvfs.analyzer.luksde_analyzer_helper.LUKSDEAnalyzerHelper* attribute), 37
- `TYPE_INDICATOR` (*dfvfs.analyzer.lvm_analyzer_helper.LVMAnalyzerHelper* attribute), 38
- `TYPE_INDICATOR` (*dfvfs.analyzer.modi_analyzer_helper.MODIANalyzerHelper* attribute), 38
- `TYPE_INDICATOR` (*dfvfs.analyzer.ntfs_analyzer_helper.NTFSAAnalyzerHelper* attribute), 39
- `TYPE_INDICATOR` (*dfvfs.analyzer.phdi_analyzer_helper.PHDIAnalyzerHelper* attribute), 39

TYPE_INDICATOR (*dfvfs.analyzer.qcow_analyzer_helper.QCOWAnalyzerHelper* attribute), 40
 TYPE_INDICATOR (*dfvfs.analyzer.qcow_analyzer_helper.QCOWAnalyzerHelper* attribute), 134
 TYPE_INDICATOR (*dfvfs.analyzer.tar_analyzer_helper.TARAnalyzerHelper* attribute), 42
 TYPE_INDICATOR (*dfvfs.analyzer.tar_analyzer_helper.TARAnalyzerHelper* attribute), 135
 TYPE_INDICATOR (*dfvfs.analyzer.tsk_analyzer_helper.TSKAnalyzerHelper* attribute), 42
 TYPE_INDICATOR (*dfvfs.analyzer.tsk_analyzer_helper.TSKAnalyzerHelper* attribute), 136
 TYPE_INDICATOR (*dfvfs.analyzer.tsk_partition_analyzer_helper.TSKPartitionAnalyzerHelper* attribute), 43
 TYPE_INDICATOR (*dfvfs.analyzer.tsk_partition_analyzer_helper.TSKPartitionAnalyzerHelper* attribute), 136
 TYPE_INDICATOR (*dfvfs.analyzer.vhdi_analyzer_helper.VHDIAnalyzerHelper* attribute), 43
 TYPE_INDICATOR (*dfvfs.analyzer.vhdi_analyzer_helper.VHDIAnalyzerHelper* attribute), 136
 TYPE_INDICATOR (*dfvfs.analyzer.vmdk_analyzer_helper.VMDKAnalyzerHelper* attribute), 44
 TYPE_INDICATOR (*dfvfs.analyzer.vmdk_analyzer_helper.VMDKAnalyzerHelper* attribute), 137
 TYPE_INDICATOR (*dfvfs.analyzer.vshadow_analyzer_helper.VShadowAnalyzerHelper* attribute), 44
 TYPE_INDICATOR (*dfvfs.analyzer.vshadow_analyzer_helper.VShadowAnalyzerHelper* attribute), 138
 TYPE_INDICATOR (*dfvfs.analyzer.xfs_analyzer_helper.XFSAnalyzerHelper* attribute), 44
 TYPE_INDICATOR (*dfvfs.analyzer.xfs_analyzer_helper.XFSAnalyzerHelper* attribute), 138
 TYPE_INDICATOR (*dfvfs.analyzer.xz_analyzer_helper.XZAnalyzerHelper* attribute), 45
 TYPE_INDICATOR (*dfvfs.analyzer.xz_analyzer_helper.XZAnalyzerHelper* attribute), 139
 TYPE_INDICATOR (*dfvfs.analyzer.zip_analyzer_helper.ZipAnalyzerHelper* attribute), 45
 TYPE_INDICATOR (*dfvfs.analyzer.zip_analyzer_helper.ZipAnalyzerHelper* attribute), 139
 TYPE_INDICATOR (*dfvfs.credentials.apfs_credentials.APFSCredentials* attribute), 49
 TYPE_INDICATOR (*dfvfs.credentials.apfs_credentials.APFSCredentials* attribute), 140
 TYPE_INDICATOR (*dfvfs.credentials.bde_credentials.BDECredentials* attribute), 49
 TYPE_INDICATOR (*dfvfs.credentials.bde_credentials.BDECredentials* attribute), 140
 type_indicator (*dfvfs.credentials.credentials.Credentials* property), 49
 type_indicator (*dfvfs.credentials.credentials.Credentials* property), 141
 TYPE_INDICATOR (*dfvfs.credentials.cs_credentials.CSCredentials* attribute), 50
 TYPE_INDICATOR (*dfvfs.credentials.cs_credentials.CSCredentials* attribute), 142
 TYPE_INDICATOR (*dfvfs.credentials.encrypted_stream_credentials.EncryptedStreamCredentials* attribute), 50
 TYPE_INDICATOR (*dfvfs.credentials.encrypted_stream_credentials.EncryptedStreamCredentials* attribute), 142
 TYPE_INDICATOR (*dfvfs.credentials.luksde_credentials.LUKSDECredentials* attribute), 51
 TYPE_INDICATOR (*dfvfs.credentials.luksde_credentials.LUKSDECredentials* attribute), 142
 type_indicator (*dfvfs.helpers.source_scanner.SourceScanner* property), 101
 type_indicator (*dfvfs.helpers.source_scanner.SourceScanner* property), 143
 TYPE_INDICATOR (*dfvfs.path.apfs_container_path_spec.APFSContainerPathSpec* attribute), 128
 TYPE_INDICATOR (*dfvfs.path.apfs_container_path_spec.APFSContainerPathSpec* attribute), 143
 TYPE_INDICATOR (*dfvfs.path.apfs_path_spec.APFSPathSpec* attribute), 129
 TYPE_INDICATOR (*dfvfs.path.apfs_path_spec.APFSPathSpec* attribute), 144
 TYPE_INDICATOR (*dfvfs.path.bde_path_spec.BDEPathSpec* attribute), 130
 TYPE_INDICATOR (*dfvfs.path.bde_path_spec.BDEPathSpec* attribute), 144
 TYPE_INDICATOR (*dfvfs.path.compressed_stream_path_spec.CompressedStreamPathSpec* attribute), 130
 TYPE_INDICATOR (*dfvfs.path.compressed_stream_path_spec.CompressedStreamPathSpec* attribute), 145
 TYPE_INDICATOR (*dfvfs.path.cpio_path_spec.CPIOPathSpec* attribute), 130
 TYPE_INDICATOR (*dfvfs.path.cpio_path_spec.CPIOPathSpec* attribute), 145
 TYPE_INDICATOR (*dfvfs.path.cs_path_spec.CSPathSpec* attribute), 131
 TYPE_INDICATOR (*dfvfs.path.cs_path_spec.CSPathSpec* attribute), 145
 TYPE_INDICATOR (*dfvfs.path.data_range_path_spec.DataRangePathSpec* attribute), 132
 TYPE_INDICATOR (*dfvfs.path.data_range_path_spec.DataRangePathSpec* attribute), 146
 TYPE_INDICATOR (*dfvfs.path.encoded_stream_path_spec.EncodedStreamPathSpec* attribute), 132
 TYPE_INDICATOR (*dfvfs.path.encoded_stream_path_spec.EncodedStreamPathSpec* attribute), 146
 TYPE_INDICATOR (*dfvfs.path.encrypted_stream_path_spec.EncryptedStreamPathSpec* attribute), 133
 TYPE_INDICATOR (*dfvfs.path.encrypted_stream_path_spec.EncryptedStreamPathSpec* attribute), 150
 TYPE_INDICATOR (*dfvfs.path.ewf_path_spec.EWFPathSpec* attribute), 133
 TYPE_INDICATOR (*dfvfs.path.ewf_path_spec.EWFPathSpec* attribute), 150
 TYPE_INDICATOR (*dfvfs.path.ext_path_spec.EXTPathSpec* attribute), 134
 TYPE_INDICATOR (*dfvfs.path.fake_path_spec.FakePathSpec* attribute), 135
 TYPE_INDICATOR (*dfvfs.path.fat_path_spec.FATPathSpec* attribute), 136
 TYPE_INDICATOR (*dfvfs.path.gpt_path_spec.GPTPathSpec* attribute), 136
 TYPE_INDICATOR (*dfvfs.path.gzip_path_spec.GzipPathSpec* attribute), 136
 TYPE_INDICATOR (*dfvfs.path.hfs_path_spec.HFSPathSpec* attribute), 137
 TYPE_INDICATOR (*dfvfs.path.luksde_path_spec.LUKSDEPathSpec* attribute), 138
 TYPE_INDICATOR (*dfvfs.path.lvm_path_spec.LVMPathSpec* attribute), 138
 TYPE_INDICATOR (*dfvfs.path.modi_path_spec.MODIPathSpec* attribute), 139
 TYPE_INDICATOR (*dfvfs.path.mount_path_spec.MountPathSpec* attribute), 139
 TYPE_INDICATOR (*dfvfs.path.ntfs_path_spec.NTFSPathSpec* attribute), 140
 TYPE_INDICATOR (*dfvfs.path.os_path_spec.OSPathSpec* attribute), 140
 type_indicator (*dfvfs.path.path_spec.PathSpec* property), 141
 TYPE_INDICATOR (*dfvfs.path.phdi_path_spec.PHDIPathSpec* attribute), 142
 TYPE_INDICATOR (*dfvfs.path.qcow_path_spec.QCOWPathSpec* attribute), 142
 TYPE_INDICATOR (*dfvfs.path.raw_path_spec.RawPathSpec* attribute), 142
 TYPE_INDICATOR (*dfvfs.path.sqlite_blob_path_spec.SQLiteBlobPathSpec* attribute), 143
 TYPE_INDICATOR (*dfvfs.path.tar_path_spec.TARPathSpec* attribute), 143
 TYPE_INDICATOR (*dfvfs.path.tsk_partition_path_spec.TSKPartitionPathSpec* attribute), 144
 TYPE_INDICATOR (*dfvfs.path.tsk_path_spec.TSKPathSpec* attribute), 144
 TYPE_INDICATOR (*dfvfs.path.vhdi_path_spec.VHDIPathSpec* attribute), 145
 TYPE_INDICATOR (*dfvfs.path.vmdk_path_spec.VMDKPathSpec* attribute), 145
 TYPE_INDICATOR (*dfvfs.path.vshadow_path_spec.VShadowPathSpec* attribute), 145
 TYPE_INDICATOR (*dfvfs.path.xfs_path_spec.XFSPathSpec* attribute), 146
 TYPE_INDICATOR (*dfvfs.path.zip_path_spec.ZipPathSpec* attribute), 146
 TYPE_INDICATOR (*dfvfs.resolver_helpers.apfs_container_resolver_helper.APFSContainerResolverHelper* attribute), 150
 TYPE_INDICATOR (*dfvfs.resolver_helpers.apfs_resolver_helper.APFSResolverHelper* attribute), 150

TYPE_INDICATOR (dfvfs.resolver_helpers.bde_resolver_helper.VHDIResolverHelper attribute), 151	TYPE_INDICATOR (dfvfs.resolver_helpers.vhdi_resolver_helper.VHDIResolverHelper attribute), 168
TYPE_INDICATOR (dfvfs.resolver_helpers.compressed_stream_resolver_helper.VMDKResolverHelper attribute), 152	TYPE_INDICATOR (dfvfs.resolver_helpers.vmdk_resolver_helper.VMDKResolverHelper attribute), 169
TYPE_INDICATOR (dfvfs.resolver_helpers.cpio_resolver_helper.VShadowResolverHelper attribute), 152	TYPE_INDICATOR (dfvfs.resolver_helpers.vshadow_resolver_helper.VShadowResolverHelper attribute), 170
TYPE_INDICATOR (dfvfs.resolver_helpers.cs_resolver_helper.XFSResolverHelper attribute), 153	TYPE_INDICATOR (dfvfs.resolver_helpers.xfs_resolver_helper.XFSResolverHelper attribute), 170
TYPE_INDICATOR (dfvfs.resolver_helpers.data_range_resolver_helper.ZipResolverHelper attribute), 154	TYPE_INDICATOR (dfvfs.resolver_helpers.zip_resolver_helper.ZipResolverHelper attribute), 171
TYPE_INDICATOR (dfvfs.resolver_helpers.encoded_stream_resolver_helper.APFSContainerFileEntry attribute), 154	TYPE_INDICATOR (dfvfs.resolver_helpers.apfs_container_file_entry.APFSContainerFileEntry attribute), 174
TYPE_INDICATOR (dfvfs.resolver_helpers.encrypted_stream_resolver_helper.APFSContainerFileEntry attribute), 155	TYPE_INDICATOR (dfvfs.resolver_helpers.apfs_container_file_entry.APFSContainerFileEntry attribute), 176
TYPE_INDICATOR (dfvfs.resolver_helpers.ewf_resolver_helper.APFSFileEntry attribute), 156	TYPE_INDICATOR (dfvfs.vfs.apfs_file_entry.APFSFileEntry attribute), 177
TYPE_INDICATOR (dfvfs.resolver_helpers.ext_resolver_helper.APFSFileSystem attribute), 156	TYPE_INDICATOR (dfvfs.vfs.apfs_file_system.APFSFileSystem attribute), 179
TYPE_INDICATOR (dfvfs.resolver_helpers.fake_resolver_helper.Attribute property), 156	TYPE_INDICATOR (dfvfs.vfs.attribute.Attribute property), 179
TYPE_INDICATOR (dfvfs.resolver_helpers.fat_resolver_helper.BDEFileEntry attribute), 157	TYPE_INDICATOR (dfvfs.vfs.bde_file_entry.BDEFileEntry attribute), 181
TYPE_INDICATOR (dfvfs.resolver_helpers.gpt_resolver_helper.BDEFileSystem attribute), 158	TYPE_INDICATOR (dfvfs.vfs.bde_file_system.BDEFileSystem attribute), 182
TYPE_INDICATOR (dfvfs.resolver_helpers.gzip_resolver_helper.CompressedStreamFileEntry attribute), 158	TYPE_INDICATOR (dfvfs.vfs.compressed_stream_file_entry.CompressedStreamFileEntry attribute), 182
TYPE_INDICATOR (dfvfs.resolver_helpers.hfs_resolver_helper.CompressedStreamFileEntry attribute), 159	TYPE_INDICATOR (dfvfs.vfs.compressed_stream_file_system.CompressedStreamFileEntry attribute), 183
TYPE_INDICATOR (dfvfs.resolver_helpers.luksde_resolver_helper.CPIOFileEntry attribute), 160	TYPE_INDICATOR (dfvfs.vfs.cpio_file_entry.CPIOFileEntry attribute), 183
TYPE_INDICATOR (dfvfs.resolver_helpers.lvm_resolver_helper.CPIOFileSystem attribute), 160	TYPE_INDICATOR (dfvfs.vfs.cpio_file_system.CPIOFileSystem attribute), 185
TYPE_INDICATOR (dfvfs.resolver_helpers.modi_resolver_helper.CSFileEntry attribute), 162	TYPE_INDICATOR (dfvfs.vfs.cs_file_entry.CSFileEntry attribute), 186
TYPE_INDICATOR (dfvfs.resolver_helpers.ntfs_resolver_helper.CSFileSystem attribute), 162	TYPE_INDICATOR (dfvfs.vfs.cs_file_system.CSFileSystem attribute), 187
TYPE_INDICATOR (dfvfs.resolver_helpers.os_resolver_helper.DataRangeFileEntry attribute), 163	TYPE_INDICATOR (dfvfs.vfs.data_range_file_entry.DataRangeFileEntry attribute), 187
TYPE_INDICATOR (dfvfs.resolver_helpers.phdi_resolver_helper.DataRangeFileSystem attribute), 163	TYPE_INDICATOR (dfvfs.vfs.data_range_file_system.DataRangeFileSystem attribute), 188
TYPE_INDICATOR (dfvfs.resolver_helpers.qcow_resolver_helper.EncodedStreamFileEntry attribute), 164	TYPE_INDICATOR (dfvfs.vfs.encoded_stream_file_entry.EncodedStreamFileEntry attribute), 189
TYPE_INDICATOR (dfvfs.resolver_helpers.raw_resolver_helper.EncodedStreamFileEntry attribute), 164	TYPE_INDICATOR (dfvfs.vfs.encoded_stream_file_system.EncodedStreamFileEntry attribute), 190
type_indicator (dfvfs.resolver_helpers.resolver_helper.ResolverHelper attribute), 165	TYPE_INDICATOR (dfvfs.vfs.encrypted_stream_file_entry.EncryptedStreamFileEntry attribute), 190
TYPE_INDICATOR (dfvfs.resolver_helpers.sqlite_blob_resolver_helper.EncryptedStreamFileEntry attribute), 166	TYPE_INDICATOR (dfvfs.vfs.encrypted_stream_file_system.EncryptedStreamFileEntry attribute), 191
TYPE_INDICATOR (dfvfs.resolver_helpers.tar_resolver_helper.EXTFileEntry attribute), 167	TYPE_INDICATOR (dfvfs.vfs.ext_file_entry.EXTFileEntry attribute), 193
TYPE_INDICATOR (dfvfs.resolver_helpers.tsk_partition_resolver_helper.EXTFileSystem attribute), 167	TYPE_INDICATOR (dfvfs.vfs.ext_file_system.EXTFileSystem attribute), 195
TYPE_INDICATOR (dfvfs.resolver_helpers.tsk_resolver_helper.FakeFileEntry attribute), 168	TYPE_INDICATOR (dfvfs.vfs.fake_file_entry.FakeFileEntry attribute), 196

TYPE_INDICATOR (*dfvfs.vfs.fake_file_system.FakeFileSystem* attribute), 199
 TYPE_INDICATOR (*dfvfs.vfs.fat_file_entry.FATFileEntry* attribute), 200
 TYPE_INDICATOR (*dfvfs.vfs.fat_file_system.FATFileSystem* attribute), 201
 type_indicator (*dfvfs.vfs.file_entry.FileEntry* property), 207
 type_indicator (*dfvfs.vfs.file_system.FileSystem* property), 210
 TYPE_INDICATOR (*dfvfs.vfs.gpt_file_entry.GPTFileEntry* attribute), 210
 TYPE_INDICATOR (*dfvfs.vfs.gpt_file_system.GPTFileSystem* attribute), 212
 TYPE_INDICATOR (*dfvfs.vfs.gzip_file_entry.GzipFileEntry* attribute), 212
 TYPE_INDICATOR (*dfvfs.vfs.gzip_file_system.GzipFileSystem* attribute), 213
 TYPE_INDICATOR (*dfvfs.vfs.hfs_file_entry.HFSFileEntry* attribute), 216
 TYPE_INDICATOR (*dfvfs.vfs.hfs_file_system.HFSFileSystem* attribute), 218
 TYPE_INDICATOR (*dfvfs.vfs.luksde_file_entry.LUKSDEFileEntry* attribute), 218
 TYPE_INDICATOR (*dfvfs.vfs.luksde_file_system.LUKSDEFileSystem* attribute), 219
 TYPE_INDICATOR (*dfvfs.vfs.lvm_file_entry.LVMFileEntry* attribute), 219
 TYPE_INDICATOR (*dfvfs.vfs.lvm_file_system.LVMFileSystem* attribute), 221
 TYPE_INDICATOR (*dfvfs.vfs.ntfs_attribute.FileNameNTFSAttribute* attribute), 221
 TYPE_INDICATOR (*dfvfs.vfs.ntfs_attribute.ObjectIdentifierNTFSAttribute* attribute), 222
 TYPE_INDICATOR (*dfvfs.vfs.ntfs_attribute.SecurityDescriptorNTFSAttribute* attribute), 222
 TYPE_INDICATOR (*dfvfs.vfs.ntfs_attribute.StandardInformationNTFSAttribute* attribute), 222
 TYPE_INDICATOR (*dfvfs.vfs.ntfs_file_entry.NTFSFileEntry* attribute), 225
 TYPE_INDICATOR (*dfvfs.vfs.ntfs_file_system.NTFSFileSystem* attribute), 227
 TYPE_INDICATOR (*dfvfs.vfs.os_file_entry.OSFileEntry* attribute), 229
 TYPE_INDICATOR (*dfvfs.vfs.os_file_system.OSFileSystem* attribute), 231
 TYPE_INDICATOR (*dfvfs.vfs.sqlite_blob_file_entry.SQLiteBlobFileEntry* attribute), 232
 TYPE_INDICATOR (*dfvfs.vfs.sqlite_blob_file_system.SQLiteBlobFileSystem* attribute), 233
 TYPE_INDICATOR (*dfvfs.vfs.tar_file_entry.TARFileEntry* attribute), 234
 TYPE_INDICATOR (*dfvfs.vfs.tar_file_system.TARFileSystem* attribute), 236
 TYPE_INDICATOR (*dfvfs.vfs.tsk_file_entry.TSKFileEntry* attribute), 239
 TYPE_INDICATOR (*dfvfs.vfs.tsk_file_system.TSKFileSystem* attribute), 242
 TYPE_INDICATOR (*dfvfs.vfs.tsk_partition_file_entry.TSKPartitionFileEntry* attribute), 243
 TYPE_INDICATOR (*dfvfs.vfs.tsk_partition_file_system.TSKPartitionFileSystem* attribute), 244
 TYPE_INDICATOR (*dfvfs.vfs.vshadow_file_entry.VShadowFileEntry* attribute), 245
 TYPE_INDICATOR (*dfvfs.vfs.vshadow_file_system.VShadowFileSystem* attribute), 247
 TYPE_INDICATOR (*dfvfs.vfs.xfs_file_entry.XFSFileEntry* attribute), 249
 TYPE_INDICATOR (*dfvfs.vfs.xfs_file_system.XFSFileSystem* attribute), 251
 TYPE_INDICATOR (*dfvfs.vfs.zip_file_entry.ZipFileEntry* attribute), 251
 TYPE_INDICATOR (*dfvfs.vfs.zip_file_system.ZipFileSystem* attribute), 253
 TYPE_INDICATOR (*dfvfs.volume.apfs_volume_system.APFSVolumeSystem* attribute), 253
 TYPE_INDICATOR (*dfvfs.volume.cs_volume_system.CSVolumeSystem* attribute), 254
 TYPE_INDICATOR (*dfvfs.volume.gpt_volume_system.GPTVolumeSystem* attribute), 255
 TYPE_INDICATOR (*dfvfs.volume.lvm_volume_system.LVMVolumeSystem* attribute), 255
 TYPE_INDICATOR (*dfvfs.volume.tsk_volume_system.TSKVolumeSystem* attribute), 255
 TYPE_INDICATOR (*dfvfs.volume.volume_system.VolumeSystem* attribute), 257
 TYPE_INDICATOR (*dfvfs.volume.vshadow_volume_system.VShadowVolumeSystem* attribute), 258
 TYPE_SOCKET (*dfvfs.vfs.attribute.StatAttribute* attribute), 180
 TYPE_WHITEOUT (*dfvfs.vfs.attribute.StatAttribute* attribute), 180

U

uncompressed_data_offset (*dfvfs.lib.zipfile.GzipMember* attribute), 21
 uncompressed_data_size (*dfvfs.file_io.gzip_file_io.GzipFile* property), 76
 uncompressed_data_size (*dfvfs.lib.zipfile.GzipCompressedStream* attribute), 119
 uncompressed_data_size

(*dfvfs.lib.zipfile.GzipMember* attribute), 122

Unlock() (*dfvfs.helpers.source_scanner.SourceScanner* method), 103

Unlock() (*dfvfs.vfs.apfs_container_file_entry.APFSContainerFileEntry* method), 175

Unlock() (*dfvfs.vfs.bde_file_entry.BDEFileEntry* method), 181

Unlock() (*dfvfs.vfs.cs_file_entry.CSFileEntry* method), 186

Unlock() (*dfvfs.vfs.file_entry.FileEntry* method), 205

Unlock() (*dfvfs.vfs.luksde_file_entry.LUKSDEFileEntry* method), 218

UnlockEncryptedVolume() (*dfvfs.helpers.command_line.CLIVolumeScannerMediator* method), 94

UnlockEncryptedVolume() (*dfvfs.helpers.volume_scanner.VolumeScannerMediator* method), 110

UnlockScanNode() (*dfvfs.helpers.source_scanner.SourceScannerContext* method), 106

unused_data (*dfvfs.compression.zlib_decompressor.ZlibDecompressor* property), 49

update_sequence_number (*dfvfs.vfs.ntfs_attribute.StandardInformationNTFSAttribute* property), 223

updated (*dfvfs.helpers.source_scanner.SourceScannerContext* attribute), 103

user_identifier (*dfvfs.lib.cpio.CPIOArchiveFileEntry* attribute), 116

UserAbort, 118

V

VHDIAnalyzerHelper (class in *dfvfs.analyzer.vhdi_analyzer_helper*), 43

VHDIFile (class in *dfvfs.file_io.vhdi_file_io*), 87

VHDIPathSpec (class in *dfvfs.path.vhdi_path_spec*), 145

VHDIResolverHelper (class in *dfvfs.resolver_helpers.vhdi_resolver_helper*), 168

VMDKAnalyzerHelper (class in *dfvfs.analyzer.vmdk_analyzer_helper*), 43

VMDKFile (class in *dfvfs.file_io.vmdk_file_io*), 88

VMDKPathSpec (class in *dfvfs.path.vmdk_path_spec*), 145

VMDKResolverHelper (class in *dfvfs.resolver_helpers.vmdk_resolver_helper*), 169

Volume (class in *dfvfs.volume.volume_system*), 256

VOLUME_IDENTIFIER_PREFIX (*dfvfs.volume.apfs_volume_system.APFSVolumeSystem* attribute), 253

VOLUME_IDENTIFIER_PREFIX (*dfvfs.volume.cs_volume_system.CSVolumeSystem* attribute), 254

VOLUME_IDENTIFIER_PREFIX (*dfvfs.volume.gpt_volume_system.GPTVolumeSystem* attribute), 255

VOLUME_IDENTIFIER_PREFIX (*dfvfs.volume.lvm_volume_system.LVMVolumeSystem* attribute), 255

VOLUME_IDENTIFIER_PREFIX (*dfvfs.volume.tsk_volume_system.TSKVolumeSystem* attribute), 255

VOLUME_IDENTIFIER_PREFIX (*dfvfs.volume.volume_system.VolumeSystem* attribute), 257

VOLUME_IDENTIFIER_PREFIX (*dfvfs.volume.vshadow_volume_system.VShadowVolumeSystem* attribute), 258

volume_identifiers (*dfvfs.volume.volume_system.VolumeSystem* property), 258

volume_index (*dfvfs.path.apfs_container_path_spec.APFSContainerPathSpec* attribute), 128

volume_index (*dfvfs.path.cs_path_spec.CSPathSpec* attribute), 131

volume_index (*dfvfs.path.lvm_path_spec.LVMPathSpec* attribute), 138

VolumeAttribute (class in *dfvfs.volume.volume_system*), 256

VolumeExtent (class in *dfvfs.volume.volume_system*), 256

volumes (*dfvfs.helpers.volume_scanner.VolumeScannerOptions* attribute), 111

volumes (*dfvfs.volume.volume_system.VolumeSystem* property), 258

VolumeScanner (class in *dfvfs.helpers.volume_scanner*), 108

VolumeScannerMediator (class in *dfvfs.helpers.volume_scanner*), 108

VolumeScannerOptions (class in *dfvfs.helpers.volume_scanner*), 110

VolumeSystem (class in *dfvfs.volume.volume_system*), 257

VolumeSystemError, 118

VShadowAnalyzerHelper (class in *dfvfs.analyzer.vshadow_analyzer_helper*), 44

VShadowDirectory (class in *dfvfs.vfs.vshadow_directory*), 245

VShadowFile (class in *dfvfs.file_io.vshadow_file_io*), 88

VShadowFileEntry (class in *dfvfs.vfs.vshadow_file_entry*), 245

VShadowFileSystem (class in *dfvfs.vfs.vshadow_file_system*), 246

VShadowPathSpec (class in *dfvfs.path.vshadow_path_spec*), 145

VShadowPathSpecGetStoreIndex() (in module *dfvfs.lib.vshadow_helper*), 127

VShadowResolverHelper (class in ZipResolverHelper (class in
dfvfs.resolver_helpers.vshadow_resolver_helper), 169
dfvfs.resolver_helpers.zip_resolver_helper), 170

VShadowVolume (class in ZlibDecompressor (class in
dfvfs.volume.vshadow_volume_system), 258
dfvfs.compression.zlib_decompressor), 48

VShadowVolumeSystem (class in
dfvfs.volume.vshadow_volume_system), 258

W

WindowsPathResolver (class in
dfvfs.helpers.windows_path_resolver), 112

WindowsVolumeScanner (class in
dfvfs.helpers.volume_scanner), 111

Write() (*dfvfs.helpers.command_line.CLIOutputWriter*
 method), 92

Write() (*dfvfs.helpers.command_line.CLITabularTableView*
 method), 92

Write() (*dfvfs.helpers.command_line.FileObjectOutputWriter*
 method), 95

Write() (*dfvfs.helpers.command_line.StdoutOutputWriter*
 method), 95

WriteSerialized() (*dfvfs.serializer.json_serializer.JsonPathSpecSerializer*
 class method), 171

WriteSerialized() (*dfvfs.serializer.serializer.PathSpecSerializer*
 method), 172

X

XFSAnalyzerHelper (class in
dfvfs.analyzer.xfs_analyzer_helper), 44

XFSDirectory (class in *dfvfs.vfs.xfs_directory*), 248

XFSExtendedAttribute (class in
dfvfs.vfs.xfs_attribute), 247

XFSFile (class in *dfvfs.file_io.xfs_file_io*), 89

XFSFileEntry (class in *dfvfs.vfs.xfs_file_entry*), 248

XFSFileSystem (class in *dfvfs.vfs.xfs_file_system*), 250

XFSPathSpec (class in *dfvfs.path.xfs_path_spec*), 146

XFSResolverHelper (class in
dfvfs.resolver_helpers.xfs_resolver_helper),
 170

XZAnalyzerHelper (class in
dfvfs.analyzer.xz_analyzer_helper), 45

XZDecompressor (class in
dfvfs.compression.xz_decompressor), 48

Z

ZipAnalyzerHelper (class in
dfvfs.analyzer.zip_analyzer_helper), 45

ZIPDirectory (class in *dfvfs.vfs.zip_directory*), 251

ZipFile (class in *dfvfs.file_io.zip_file_io*), 90

ZipFileEntry (class in *dfvfs.vfs.zip_file_entry*), 251

ZipFileSystem (class in *dfvfs.vfs.zip_file_system*), 252

ZipPathSpec (class in *dfvfs.path.zip_path_spec*), 146