Digital Forensics Artifacts Repository

Release 20221121

unknown

Dec 05, 2022

CONTENTS

Digital Forensics Artifacts Repository, is a free, community-sourced, machine-readable knowledge base of digital forensic artifacts that the world can use both as an information source and within other tools.

The source code is available from the project page.

CHAPTER

ONE

GETTING STARTED

To be able to use Forensics Artifacts you first need to install it. There are multiple ways to install Forensics Artifacts, 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 artifactsenv
cd artifactsenv
source ./bin/activate
```

Upgrade pip and install Forensics Artifacts dependencies:

```
pip install --upgrade pip
pip install artifacts
```

To deactivate the virtualenv run:

```
deactivate
```

1.1.2 Ubuntu 18.04 and 20.04 LTS

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

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

Update and install Forensics Artifacts:

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

1.1.3 Windows

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

- main; branch intended for the "packaged release" of Forensics Artifacts and dependencies;
- dev; branch intended for the "development release" of Forensics Artifacts;
- 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:\Python38\python.exe tools\update.py --preset artifacts
```

CHAPTER

TWO

BACKGROUND

The first version of the artifact definitions originated from the GRR project, where it is used to describe and quickly collect data of interest, for example specific files or Windows Registry keys. The goal of the format is to provide a tool independent way to describe the majority of forensic artifacts in a language that is readable by humans and machines.

The format is designed to be simple and straight forward, so that a digital forensic analysist is able to quickly write artifact definitions during an investigation without having to rely on complex standards or tooling.

The format is intended to describe forensically-relevant data on a machine, while being tool agnostic. In particular we intentionally avoided adding IOC-like logic, or describing how the data should be collected since this various between tools.

For some background on the artifacts system and how we expect it to be used see this Blackhat presentation and YouTube video from the GRR team.

2.1 Terminology

The term artifact (or artefact) is widely used within computer (or digital) forensics, though there is no official definition of this term.

The definition closest to the meaning of the word within computer forensics is that of the word artifact within archaeology. The term should not be confused with the word artifact used within software development.

If archaeology defines an artifact as:

```
something made or given shape by man, such as a tool or
a work of art, esp an object of archaeological interest
```

The definition of artifact within computer forensics could be:

```
An object of digital archaeological interest.
```

Where digital archaeology roughly refers to computer forensics without the forensic (legal) context.

2.2 Statistics

The artifact definitions can be found in the data directory and the format is described in detail in the Style Guide.

Status of the repository as of 2022-11-21

2.2.1 Artifact definition source types

2.2.2 Operating systems

CHAPTER

THREE

ARTIFACT DEFINITION FORMAT AND STYLE GUIDE

The best way to show what an artifact definition is, is by example. The following example is the artifact definition for the Windows EVTX System Event Logs.

```
name: WindowsSystemEventLogEvtx
doc: Windows System Event log for Vista or later systems.
sources:
- type: FILE
  attributes: {paths: ['%%environ_systemroot%%\System32\winevt\Logs\System.evtx']}
supported_os: [Windows]
urls: ['https://artifacts-kb.readthedocs.io/en/latest/sources/windows/EventLog.html']
```

The artifact definition can have the following values:

3.1 Deprecated values

3.2 Name

The name of an artifact definition should be in CamelCase name without spaces.

Prefix platform specific artifact definitions with the name of the operating system using “Linux”, “MacOS” or “Windows”.

If not platform specific:

- prefix with the application name, for example “ChromeHistory”.
- prefix with the name of the subsystem, for example “WMIComputerSystemProduct”.

Commonly used prefixes:

Suffix artifact definitions with the type of artifact, for example are files use “BrowserHistoryFile” instead of “BrowserHistory” to reduce ambiguity.

3.3 Description

Style note: Typically one line description of the artifact, mentioning important caveats. If more than one line is necessary, use the multi-line YAML Literal Style as indicated by the | character.

```
doc: |
  The Windows run keys.

  Note users.sid will currently only expand to SIDs with profiles on the system,
  not all SIDs.
```

Style note: the short description (first line) and the longer portion are separated by an empty line.

Style note: explicit newlines (\n) should not be used.

3.4 Sources

Every source definition starts with a `type` followed by arguments for example:

```
sources:
- type: COMMAND
  attributes:
    args: [-qa]
    cmd: /bin/rpm
```

```
sources:
- type: FILE
  attributes:
    paths:
    - /root/.bashrc
    - /root/.cshrc
    - /root/.ksh
    - /root/.logout
    - /root/.profile
    - /root/.tcsh
    - /root/.zlogin
    - /root/.zlogout
    - /root/.zprofile
    - /root/.zprofile
```

Style note: where sources take a single argument with a single value, the one-line {} form should be used to save on line breaks as below:

```
- type: FILE
  attributes: {paths: ['%%environ_systemroot%%\System32\winevt\Logs\System.evtx']}
```

3.5 Deprecated values

3.5.1 Source types

Currently the following different source types are defined:

The sources types are defined in definitions.py. as TYPE_INDICATOR constants.

3.5.2 Artifact group source

The artifact group source is a source that consists of a group of other artifacts e.g.

```
- type: ARTIFACT_GROUP
  attributes:
    names:
    - WindowsRunKeys
    - WindowsServices
```

Where `attributes` can contain the following values:

3.5.3 Command source

The command source is a source that consists of the output of a command e.g.

```
- type: COMMAND
  attributes:
    args: [-qa]
    cmd: /bin/rpm
```

Where `attributes` can contain the following values:

3.5.4 File source

The file source is a source that consists of the contents of files e.g.

```
- type: FILE
  attributes:
    paths: ['%%environ_systemroot%%\System32\winevt\Logs\System.evtx']
```

Where `attributes` can contain the following values:

3.5.5 Path source

The path source is a source that consists of the contents of paths e.g.

```
- type: PATH
  attributes:
    paths: ['\Program Files']
    separator: '\'
```

Where `attributes` can contain the following values:

3.5.6 Windows Registry key source

The Windows Registry key source is a source that consists of the contents of Windows Registry keys e.g.

```
sources:
- type: REGISTRY_KEY
  attributes:
    keys:
    - 'HKEY_USERS\%%users.sid%%\Software\Microsoft\Internet Explorer\TypedURLs\*'
```

Where `attributes` can contain the following values:

3.5.7 Windows Registry value source

The Windows Registry value source is a source that consists of the contents of Windows Registry values e.g.

```
- type: REGISTRY_VALUE
  attributes:
    key_value_pairs:
    - {key: 'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\
↪WindowsUpdate', value: 'CISCNF4654'}
```

Where `attributes` can contain the following values:

3.5.8 Windows Management Instrumentation (WMI) query source

The Windows Management Instrumentation (WMI) query source is a source that consists of the output of Windows Management Instrumentation (WMI) queries e.g.

```
- type: WMI
  attributes:
    query: SELECT * FROM Win32_UserAccount WHERE name='%%users.username%%'
```

Where `attributes` can contain the following values:

3.6 Supported operating system

Since operating system (OS) are a very common constraint, this has been provided as a separate option "supported_os" to simplify syntax. For supported_os no quotes are required. The currently supported operating systems are:

- Darwin (also used for Mac OS X)
- Linux
- Windows

```
supported_os: [Darwin, Linux, Windows]
```

This can be translated to objectfilter as:

```
["os =='Darwin'" OR "os=='Linux'" OR "os == 'Windows'"]
```

3.7 Parameter expansion and globs

Artifact definitions can use different types of parameters that need to be expanded at runtime, such as:

- POSIX users variables, for example %%users.homedir%%
- Windows evironment variables, for example %%environ_systemroot%%
- Windows users variables, for example %%users.temp%%

3.7.1 POSIX users variables

Supported POSIX users variables are:

Decomposition rules

Note that the following decomposition rules are approximations based on common usage scenarios.

%%users.homedir%% can be decomposed into:

- '/Users/*' for Mac OS
- '/home/*' and '/root' for Linux

3.7.2 Windows evironment variables

Supported Windows evironment variables are:

3.7.3 Windows users variables

Supported Windows users variables are:

Decomposition rules

TODO: add information about system accounts

Note that the following decomposition rules are approximations based on common usage scenarios.

%%users.appdata%% can be decomposed into:

- '%%users.userprofile%%\AppData\Roaming' for Windows Vista and later
- '%%users.userprofile%%\Application Data'

%%users.localappdata%% can be decomposed into:

- '%%users.userprofile%%\AppData\Local' for Windows Vista and later
- '%%users.userprofile%%\Local Settings\Application Data'

%%users.localappdata_low%% can be decomposed into:

- '%%users.userprofile%%\AppData\LocalLow' for Windows Vista and later

%%users.temp%% can be decomposed into:

- '%%users.localappdata%%\Temp'

%%users.userprofile%% can be decomposed into:

- 'Documents and Settings*'
- 'Users*' for Windows Vista and later

3.8 Additional style notes

3.8.1 Artifact definition YAML files

Artifact definition YAML filenames should be of the form:

```
$FILENAME.yaml
```

Where $FILENAME is name of the file e.g. windows.yaml.

Each definition file should have a comment at the top of the file with a one-line summary describing the type of artifact definitions contained in the file e.g.

```
# Windows specific artifacts.
```

3.8.2 Lists

Generally use the short [] format for single-item lists that fit inside 80 characters to save on unnecessary line breaks:

```
supported_os: [Windows]
urls: ['https://artifacts-kb.readthedocs.io/en/latest/sources/windows/EventLog.html']
```

and the bulleted list form for multi-item lists or long lines:

```
paths:
- 'HKEY_USERS\%%users.sid%%\Software\Microsoft\Windows\CurrentVersion\Run\*'
- 'HKEY_USERS\%%users.sid%%\Software\Microsoft\Windows\CurrentVersion\RunOnce\*'
- 'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\*'
- 'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce\*'
- 'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnceEx\*'
```

3.8.3 Quotes

Quotes should not be used for doc strings, artifact names, and simple lists like supported_os.

Paths and URLs should use single quotes to avoid the need for manual escaping.

```
paths: ['%%environ_temp%%\*.exe']
urls: ['https://artifacts-kb.readthedocs.io/en/latest/sources/windows/EventLog.html']
```

Double quotes should be used where escaping causes problems, such as regular expressions:

```
content_regex_list: ["^%%users.username%%:[^:]*\n"]
```

3.8.4 Minimize the number of definitions by using multiple sources

To minimize the number of artifacts in the list, combine them using the supported_os attributes where it makes sense. e.g. rather than having FirefoxHistoryWindows, FirefoxHistoryLinux, FirefoxHistoryDarwin, do:

```
name: FirefoxHistory
doc: Firefox places.sqlite files.
sources:
- type: FILE
  attributes:
    paths:
    - %%users.localappdata%%\Mozilla\Firefox\Profiles\*\places.sqlite
    - %%users.appdata%%\Mozilla\Firefox\Profiles\*\places.sqlite
  supported_os: [Windows]
- type: FILE
  attributes:
    paths: [%%users.homedir%%/Library/Application Support/Firefox/Profiles/*/places.
↪sqlite]
  supported_os: [Darwin]
- type: FILE
  attributes:
    paths: ['%%users.homedir%%/.mozilla/firefox/*/places.sqlite']
  supported_os: [Linux]
supported_os: [Windows, Linux, Darwin]
```

CHAPTER

FOUR

ARTIFACTS PACKAGE

4.1 Submodules

4.2 artifacts.artifact module

The artifact definition.

class artifacts.artifact.**ArtifactDefinition**(*name*, *aliases=None*, *description=None*)

Bases: object

Artifact definition interface.

aliases

aliases that identify the artifact definition.

Type
list[str]

description

description.

Type
str

name

name that uniquely identifiers the artifact definition.

Type
str

provides

hints to what information the artifact definition provides.

Type
list[str]

sources

sources.

Type
list[str]

supported_os

supported operating systems.

Type
list[str]

urls

URLs with more information about the artifact definition.

Type
list[str]

AppendSource(*type_indicator*, *attributes*)

Appends a source.

If you want to implement your own source type you should create a subclass in source_type.py and change the AppendSource method to handle the new subclass. This function raises FormatError if an unsupported source type indicator is encountered.

Parameters

- **type_indicator** (*str*) – source type indicator.
- **attributes** (*dict[str, object]*) – source attributes.

Returns
a source type.

Return type
SourceType

Raises
FormatError – if the type indicator is not set or unsupported, or if required attributes are missing.

AsDict()

Represents an artifact as a dictionary.

Returns
artifact attributes.

Return type
dict[str, object]

4.3 artifacts.definitions module

Constants and definitions.

4.4 artifacts.errors module

The error objects.

exception artifacts.errors.**CodeStyleError**

Bases: *Error*

Error that is raised when code formatting fails style checks.

exception artifacts.errors.**Error**

Bases: Exception

The error interface.

exception artifacts.errors.**FormatError**

Bases: *Error*

Error that is raised when the format is incorrect.

exception artifacts.errors.**MissingDependencyError**

Bases: *Error*

Artifact references artifact that is undefined.

4.5 artifacts.reader module

The artifact reader objects.

class artifacts.reader.**ArtifactsReader**

Bases: *BaseArtifactsReader*

Artifacts reader common functionality.

ReadArtifactDefinitionValues(*artifact_definition_values*)

Reads an artifact definition from a dictionary.

Parameters
artifact_definition_values (*dict[str, object]*) – artifact definition values.

Returns
an artifact definition.

Return type
ArtifactDefinition

Raises
FormatError – if the format of the artifact definition is not set or incorrect.

ReadDirectory(*path*, *extension='yaml'*)

Reads artifact definitions from a directory.

This function does not recurse sub directories.

Parameters

- **path** (*str*) – path of the directory to read from.
- **extension** (*Optional[str]*) – extension of the filenames to read.

Yields
ArtifactDefinition – an artifact definition.

ReadFile(*filename*)

Reads artifact definitions from a file.

Parameters
filename (*str*) – name of the file to read from.

Yields
ArtifactDefinition – an artifact definition.

abstract ReadFileObject(*file_object*)

Reads artifact definitions from a file-like object.

Parameters
file_object (*file*) – file-like object to read from.

Yields
ArtifactDefinition – an artifact definition.

Raises
FormatError – if the format of the artifact definition is not set or incorrect.

class artifacts.reader.**BaseArtifactsReader**

Bases: object

Artifacts reader interface.

supported_os

supported operating systems.

Type
set[str]

abstract ReadArtifactDefinitionValues(*artifact_definition_values*)

Reads an artifact definition from a dictionary.

Parameters
artifact_definition_values (*dict[str, object]*) – artifact definition values.

Returns
an artifact definition.

Return type
ArtifactDefinition

Raises
FormatError – if the format of the artifact definition is not set or incorrect.

abstract ReadDirectory(*path*, *extension=None*)

Reads artifact definitions from a directory.

This function does not recurse sub directories.

Parameters

- **path** (*str*) – path of the directory to read from.
- **extension** (*Optional[str]*) – extension of the filenames to read.

Yields
ArtifactDefinition – an artifact definition.

abstract ReadFile(*filename*)

Reads artifact definitions from a file.

Parameters
filename (*str*) – name of the file to read from.

Yields
ArtifactDefinition – an artifact definition.

abstract ReadFileObject(*file_object*)

Reads artifact definitions from a file-like object.

Parameters
file_object (*file*) – file-like object to read from.

Yields
ArtifactDefinition – an artifact definition.

Raises
FormatError – if the format of the artifact definition is not set or incorrect.

class artifacts.reader.**JsonArtifactsReader**

Bases: *ArtifactsReader*

JSON artifacts reader.

ReadFileObject(*file_object*)

Reads artifact definitions from a file-like object.

Parameters
file_object (*file*) – file-like object to read from.

Yields
ArtifactDefinition – an artifact definition.

Raises
FormatError – if the format of the JSON artifact definition is not set or incorrect.

class artifacts.reader.**YamlArtifactsReader**

Bases: *ArtifactsReader*

YAML artifacts reader.

ReadFileObject(*file_object*)

Reads artifact definitions from a file-like object.

Parameters
file_object (*file*) – file-like object to read from.

Yields
ArtifactDefinition – an artifact definition.

Raises
FormatError – if the format of the YAML artifact definition is not set or incorrect.

4.6 artifacts.registry module

The artifact definitions registry.

class artifacts.registry.**ArtifactDefinitionsRegistry**

Bases: object

Artifact definitions registry.

classmethod CreateSourceType(*type_indicator*, *attributes*)

Creates a source type object.

Parameters
- **type_indicator** (*str*) – source type indicator.
- **attributes** (*dict[str, object]*) – source attributes.

Returns
a source type.

Return type
SourceType

Raises
FormatError – if the type indicator is not set or unsupported, or if required attributes are missing.

`DeregisterDefinition`(*artifact_definition*)

Deregisters an artifact definition.

Artifact definitions are identified based on their lower case name.

Parameters
artifact_definition (ArtifactDefinition) – an artifact definition.

Raises
KeyError – if an artifact definition is not set for the corresponding name.

`classmethod DeregisterSourceType`(*source_type_class*)

Deregisters a source type.

Source types are identified based on their type indicator.

Parameters
source_type_class (*type*) – source type.

Raises
KeyError – if a source type is not set for the corresponding type indicator.

`GetDefinitionByAlias`(*alias*)

Retrieves a specific artifact definition by alias.

Parameters
alias (*str*) – alias of the artifact definition.

Returns
an artifact definition or None if not available.

Return type
ArtifactDefinition

`GetDefinitionByName`(*name*)

Retrieves a specific artifact definition by name.

Parameters
name (*str*) – name of the artifact definition.

Returns
an artifact definition or None if not available.

Return type
ArtifactDefinition

`GetDefinitions`()

Retrieves the artifact definitions.

Returns
artifact definitions.

Return type
list[*ArtifactDefinition*]

GetUndefinedArtifacts()

Retrieves the names of undefined artifacts used by artifact groups.

Returns
undefined artifacts names.

Return type
set[str]

ReadFileObject(*artifacts_reader*, *file_object*)

Reads artifact definitions into the registry from a file-like object.

Parameters

- **artifacts_reader** (ArtifactsReader) – an artifacts reader.
- **file_object** (*file*) – file-like object to read from.

ReadFromDirectory(*artifacts_reader*, *path*, *extension='yaml'*)

Reads artifact definitions into the registry from files in a directory.

This function does not recurse sub directories.

Parameters

- **artifacts_reader** (ArtifactsReader) – an artifacts reader.
- **path** (*str*) – path of the directory to read from.
- **extension** (*Optional[str]*) – extension of the filenames to read.

Raises
KeyError – if a duplicate artifact definition is encountered.

ReadFromFile(*artifacts_reader*, *filename*)

Reads artifact definitions into the registry from a file.

Parameters

- **artifacts_reader** (ArtifactsReader) – an artifacts reader.
- **filename** (*str*) – name of the file to read from.

RegisterDefinition(*artifact_definition*)

Registers an artifact definition.

Artifact definitions are identified based on their lower case name.

Parameters
artifact_definition (ArtifactDefinition) – an artifact definition.

Raises
KeyError – if artifact definition is already set for the corresponding name or alias.

classmethod RegisterSourceType(*source_type_class*)

Registers a source type.

Source types are identified based on their type indicator.

Parameters
source_type_class (*type*) – source type.

Raises
KeyError – if source types is already set for the corresponding type indicator.

classmethod RegisterSourceTypes(*source_type_classes*)

Registers source types.

Source types are identified based on their type indicator.

Parameters
source_type_classes (*list[type]*) – source types.

4.7 artifacts.source_type module

The source type objects.

The source type objects define the source of the artifact data. In earlier versions of the artifact definitions collector definitions had a similar purpose as the source type. Currently the following source types are defined: * artifact; the source is one or more artifact definitions; * file; the source is one or more files; * path; the source is one or more paths; * Windows Registry key; the source is one or more Windows Registry keys; * Windows Registry value; the source is one or more Windows Registry values; * WMI query; the source is a Windows Management Instrumentation query.

The difference between the file and path source types are that file should be used to define file entries that contain data and path, file entries that define a location. E.g. on Windows %SystemRoot% could be considered a path artifact definition, pointing to a location e.g. C:Windows. And where C:WindowsSystem32winevtLogsAppEvent.evt a file artifact definition, pointing to the Application Event Log file.

class artifacts.source_type.**ArtifactGroupSourceType**(*names=None*)

Bases: *SourceType*

Artifact group source type.

AsDict()

Represents a source type as a dictionary.

Returns
source type attributes.

Return type
dict[str, str]

TYPE_INDICATOR = 'ARTIFACT_GROUP'

class artifacts.source_type.**CommandSourceType**(*args=None*, *cmd=None*)

Bases: *SourceType*

Command source type.

AsDict()

Represents a source type as a dictionary.

Returns
source type attributes.

Return type
dict[str, str]

TYPE_INDICATOR = 'COMMAND'

class artifacts.source_type.**DirectorySourceType**(*paths=None*, *separator='/'*)

Bases: *SourceType*

Directory source type.

AsDict()

Represents a source type as a dictionary.

Returns
source type attributes.

Return type
dict[str, str]

TYPE_INDICATOR = 'DIRECTORY'

class artifacts.source_type.**FileSourceType**(*paths=None*, *separator='/'*)

Bases: *SourceType*

File source type.

AsDict()

Represents a source type as a dictionary.

Returns
source type attributes.

Return type
dict[str, str]

TYPE_INDICATOR = 'FILE'

class artifacts.source_type.**PathSourceType**(*paths=None*, *separator='/'*)

Bases: *SourceType*

Path source type.

AsDict()

Represents a source type as a dictionary.

Returns
source type attributes.

Return type
dict[str, str]

TYPE_INDICATOR = 'PATH'

class artifacts.source_type.**SourceType**

Bases: object

Artifact definition source type interface.

abstract AsDict()

Represents a source type as a dictionary.

Returns
source type attributes.

Return type
dict[str, str]

TYPE_INDICATOR = None

property type_indicator

type indicator.

Type
str

class artifacts.source_type.SourceTypeFactory

Bases: object

Source type factory.

classmethod CreateSourceType(*type_indicator*, *attributes*)

Creates a source type.

Parameters

- **type_indicator** (*str*) – source type indicator.
- **attributes** (*dict[str, object]*) – source type attributes.

Returns
a source type.

Return type
SourceType

Raises
FormatError – if the type indicator is not set or unsupported, or if required attributes are missing.

classmethod DeregisterSourceType(*source_type_class*)

Deregisters a source type.

Source types are identified based on their type indicator.

Parameters
source_type_class (*type*) – source type.

Raises
KeyError – if a source type is not set for the corresponding type indicator.

classmethod GetSourceTypeIndicators()

Retrieves the source type indicators.

Returns
source type indicators.

Return type
list[str]

classmethod GetSourceTypes()

Retrieves the source types.

Returns
source types.

Return type
list[type]

classmethod RegisterSourceType(*source_type_class*)

Registers a source type.

Source types are identified based on their type indicator.

Parameters
source_type_class (*type*) – source type.

Raises
KeyError – if source types is already set for the corresponding type indicator.

classmethod RegisterSourceTypes(*source_type_classes*)

Registers source types.

Source types are identified based on their type indicator.

Parameters
source_type_classes (*list[type]*) – source types.

class artifacts.source_type.**WMIQuerySourceType**(*base_object=None*, *query=None*)

Bases: *SourceType*

WMI query source type.

base_object

WMI base object.

Type
str

query

WMI query.

Type
str

AsDict()

Represents a source type as a dictionary.

Returns
source type attributes.

Return type
dict[str, str]

TYPE_INDICATOR = 'WMI'

class artifacts.source_type.**WindowsRegistryKeySourceType**(*keys=None*)

Bases: *SourceType*

Windows Registry key source type.

AsDict()

Represents a source type as a dictionary.

Returns
source type attributes.

Return type
dict[str, str]

TYPE_INDICATOR = 'REGISTRY_KEY'

VALID_PREFIXES = ['HKEY_LOCAL_MACHINE', 'HKEY_USERS', 'HKEY_CLASSES_ROOT', '%%current_control_set%%']

classmethod ValidateKey(*key_path*)

Validates this key against supported key names.

Parameters
key_path (*str*) – path of a Windows Registry key.

Raises
FormatError – when key is not supported.

class artifacts.source_type.**WindowsRegistryValueSourceType**(*key_value_pairs=None*)

Bases: *SourceType*

Windows Registry value source type.

AsDict()

Represents a source type as a dictionary.

Returns
source type attributes.

Return type
dict[str, str]

TYPE_INDICATOR = 'REGISTRY_VALUE'

4.8 artifacts.writer module

The artifact writer objects.

class artifacts.writer.**ArtifactWriter**

Bases: *BaseArtifactsWriter*

File artifacts writer.

abstract FormatArtifacts(*artifacts*)

Formats artifacts to desired output format.

Parameters
artifacts (ArtifactDefinition|*list[*ArtifactDefinition*]*) – artifact definitions.

Returns
formatted string of artifact definition.

Return type
str

WriteArtifactsFile(*artifacts*, *filename*)

Writes artifact definitions to a file.

Parameters

- **artifacts** (*list[*ArtifactDefinition*]*) – artifact definitions to be written.
- **filename** (*str*) – name of the file to write artifacts to.

class artifacts.writer.**BaseArtifactsWriter**

Bases: object

Artifacts writer interface.

abstract FormatArtifacts(*artifacts*)

Formats artifacts to desired output format.

Parameters
artifacts (*list[ArtifactDefinition]*) – artifact definitions.

Returns
formatted string of artifact definition.

Return type
str

abstract WriteArtifactsFile(*artifacts*, *filename*)

Writes artifact definitions to a file.

Parameters

- **artifacts** (*list[ArtifactDefinition]*) – artifact definitions to be written.
- **filename** (*str*) – name of the file to write artifacts to.

class artifacts.writer.**JsonArtifactsWriter**

Bases: *ArtifactWriter*

JSON artifacts writer interface.

FormatArtifacts(*artifacts*)

Formats artifacts to desired output format.

Parameters
artifacts (*list[ArtifactDefinition]*) – artifact definitions.

Returns
formatted string of artifact definition.

Return type
str

class artifacts.writer.**YamlArtifactsWriter**

Bases: *ArtifactWriter*

YAML artifacts writer interface.

FormatArtifacts(*artifacts*)

Formats artifacts to desired output format.

Parameters
artifacts (*list[ArtifactDefinition]*) – artifact definitions.

Returns
formatted string of artifact definition.

Return type
str

4.9 Module contents

ForensicArtifacts.com Artifact Repository.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

a

INDEX

G

J

M

N

P

Q

R

S

T

U

V

W

Y