
Volatility 3 Documentation

Release 2.5.2

Volatility Foundation

Jan 31, 2024

DOCUMENTATION

1	Volatility 3 Basics	3
1.1	Memory layers	3
1.2	Templates and Objects	4
1.3	Symbol Tables	5
1.4	Plugins	5
1.5	Output Renderers	5
1.6	Configuration Tree	6
1.7	Automagic	6
2	Writing Plugins	7
2.1	How to Write a Simple Plugin	7
2.2	Writing more advanced Plugins	12
2.3	Using Volatility 3 as a Library	17
3	Creating New Symbol Tables	23
3.1	How Volatility finds symbol tables	23
3.2	Windows symbol tables	23
3.3	Mac or Linux symbol tables	24
4	Changes between Volatility 2 and Volatility 3	25
4.1	Library and Context	25
4.2	Symbols and Types	25
4.3	Object Model changes	25
4.4	Layer and Layer dependencies	26
4.5	Automagic	26
4.6	Searching and Scanning	26
4.7	Output Rendering	26
5	Volshell - A CLI tool for working with memory	27
5.1	Starting volshell	27
5.2	Accessing objects	28
5.3	Running plugins	29
5.4	Running scripts	30
5.5	Loading files	30
6	Glossary	31
6.1	A	31
6.2	D	31
6.3	M	32
6.4	O	32
6.5	P	32

6.6	R	33
6.7	S	33
6.8	T	33
6.9	U	34
7	Linux Tutorial	35
7.1	Acquiring memory	35
7.2	Procedure to create symbol tables for linux	35
7.3	Listing plugins	35
7.4	Using plugins	36
7.5	Example	36
8	macOS Tutorial	39
8.1	Acquiring memory	39
8.2	Procedure to create symbol tables for macOS	39
8.3	Listing plugins	39
8.4	Using plugins	40
8.5	Example	40
9	Windows Tutorial	43
9.1	Acquiring memory	43
9.2	Listing Plugins	43
9.3	Using plugins	44
9.4	Example	44
10	volatility3 package	47
10.1	Subpackages	48
11	Indices and tables	769
	Python Module Index	771
	Index	775

This is the documentation for Volatility 3, the most advanced memory forensics framework in the world. Like previous versions of the Volatility framework, Volatility 3 is Open Source.

List of plugins

Below is the main documentation regarding volatility 3:

VOLATILITY 3 BASICS

Volatility splits memory analysis down to several components. The main ones are:

- Memory layers
- Templates and Objects
- Symbol Tables

Volatility 3 stores all of these within a *Context*, which acts as a container for all the various layers and tables necessary to conduct memory analysis.

1.1 Memory layers

A memory layer is a body of data that can be accessed by requesting data at a specific address. At its lowest level this data is stored on a physical medium (RAM) and very early computers addresses locations in memory directly. However, as the size of memory increased and it became more difficult to manage memory most architectures moved to a “paged” model of memory, where the available memory is cut into specific fixed-sized pages. To help further, programs can ask for any address and the processor will look up their (virtual) address in a map, to find out where the (physical) address that it lives at is, in the actual memory of the system.

Volatility can work with these layers as long as it knows the map (so, for example that virtual address 1 looks up at physical address 9). The automagic that runs at the start of every volatility session often locates the kernel’s memory map, and creates a kernel virtual layer, which allows for kernel addresses to be looked up and the correct data returned. There can, however, be several maps, and in general there is a different map for each process (although a portion of the operating system’s memory is usually mapped to the same location across all processes). The maps may take the same address but point to a different part of physical memory. It also means that two processes could theoretically share memory, but having an virtual address mapped to the same physical address as another process. See the worked example below for more information.

To translate an address on a layer, call *layer.mapping(offset, length, ignore_errors)* and it will return a list of chunks without overlap, in order, for the requested range. If a portion cannot be mapped, an exception will be thrown unless *ignore_errors* is true. Each chunk will contain the original offset of the chunk, the translated offset, the original size and the translated size of the chunk, as well as the lower layer the chunk lives within.

1.1.1 Worked example

The operating system and two programs may all appear to have access to all of physical memory, but actually the maps they each have mean they each see something different:

Listing 1: Memory mapping example

Operating system map	Physical Memory
1 -> 9	1 - Free
2 -> 3	2 - OS.4, Process 1.4, Process 2.4
3 -> 7	3 - OS.2
4 -> 2	4 - Free
	5 - Free
Process 1 map	6 - Process 1.2, Process 2.3
1 -> 12	7 - OS.3
2 -> 6	8 - Process1.3
3 -> 8	9 - OS.1
4 -> 2	10 - Process2.1
	11 - Free
Process 2 map	12 - Process1.1
1 -> 10	13 - Free
2 -> 15	14 - Free
3 -> 6	15 - Process2.2
4 -> 2	16 - Free

In this example, part of the operating system is visible across all processes (although not all processes can write to the memory, there is a permissions model for intel addressing which is not discussed further here).)

In Volatility 3 mappings are represented by a directed graph of layers, whose end nodes are *DataLayers* and whose internal nodes are *TranslationLayers*. In this way, a raw memory image in the LiME file format and a page file can be combined to form a single Intel virtual memory layer. When requesting addresses from the Intel layer, it will use the Intel memory mapping algorithm, along with the address of the directory table base or page table map, to translate that address into a physical address, which will then either be directed towards the swap layer or the LiME layer. Should it be directed towards the LiME layer, the LiME file format algorithm will be translate the new address to determine where within the file the data is stored. When the *layer.read()* method is called, the translation is done automatically and the correct data gathered and combined.

Note: Volatility 2 had a similar concept, called address spaces, but these could only stack linearly one on top of another.

The list of layers supported by volatility can be determined by running the *frameworkinfo* plugin.

1.2 Templates and Objects

Once we can address contiguous chunks of memory with a means to translate a virtual address (as seen by the programs) into the actual data used by the processor, we can start pulling out *Objects* by taking a *Template* and constructing it on the memory layer at a specific offset. A *Template* contains all the information you can know about the structure of the object without actually being populated by any data. As such a *Template* can tell you the size of a structure and its members, how far into the structure a particular member lives and potentially what various values in that field would mean, but not what resides in a particular member.

Using a *Template* on a memory layer at a particular offset, an *Object* can be constructed. In Volatility 3, once an *Object* has been created, the data has been read from the layer and is not read again. An object allows its members

to be interrogated and in particular allows pointers to be followed, providing easy access to the data contained in the object.

Note: Volatility 2 would re-read the data which was useful for live memory forensics but quite inefficient for the more common static memory analysis typically conducted. Volatility 3 requires that objects be manually reconstructed if the data may have changed. Volatility 3 also constructs actual Python integers and floats whereas Volatility 2 created proxy objects which would sometimes cause problems with type checking.

1.3 Symbol Tables

Most compiled programs know of their own templates, and define the structure (and location within the program) of these templates as a *Symbol*. A *Symbol* is often an address and a template and can be used to refer to either independently. Lookup tables of these symbols are often produced as debugging information alongside the compilation of the program. Volatility 3 provides access to these through a *SymbolTable*, many of which can be collected within a *Context* as a *SymbolSpace*. A *Context* can store only one *SymbolSpace* at a time, although a *SymbolSpace* can store as many *SymbolTable* items as necessary.

Volatility 3 uses the de facto naming convention for symbols of *module!symbol* to refer to them. It reads them from its own JSON formatted file, which acts as a common intermediary between Windows PDB files, Linux DWARF files, other symbol formats and the internal Python format that Volatility 3 uses to represent a *Template* or a *Symbol*.

Note: Volatility 2's name for a *SymbolSpace* was a profile, but it could not differentiate between symbols from different modules and required special handling for 32-bit programs that used Wow64 on Windows. This meant that all symbols lived in a single namespace with the possibility of symbol name collisions. It read the symbols using a format called *vtypes*, written in Python code directly. This made it less transferable or able to be used by other software.

1.4 Plugins

A plugin acts as a means of requesting data from the user interface (and so the user) and then using it to carry out a specific form of analysis on the *Context* (containing whatever symbol tables and memory layers it may). The means of communication between the user interface and the library is the configuration tree, which is used by components within the *Context* to store configurable data. After the plugin has been run, it then returns the results in a specific format known as a *TreeGrid*. This ensures that the data can be handled by consumers of the library, without knowing exactly what the data is or how it's formatted.

1.5 Output Renderers

User interfaces can choose how best to present the output of the results to their users. The library always responds from every plugin with a *TreeGrid*, and the user interface can then determine how best to display it. For the Command Line Interface, that might be via text output as a table, or it might output to an SQLite database or a CSV file. For a web interface, the best output is probably as JSON where it could be displayed as a table, or inserted into a database like Elastic Search and trawled using an existing frontend such as Kibana.

The renderers only need to know how to process very basic types (booleans, strings, integers, bytes) and a few additional specific ones (disassembly and various absent values).

1.6 Configuration Tree

The configuration tree acts as the interface between the calling program and Volatility 3 library. Elements of the library (such as a *Plugin*, a *TranslationLayer*, an *Automagic*, etc.) can use the configuration tree to inform the calling program of the options they require and/or optionally support, and allows the calling program to provide that information when the library is then called.

1.7 Automagic

There are certain setup tasks that establish the context in a way favorable to a plugin before it runs, removing several tasks that are repetitive and also easy to get wrong. These are called *Automagic*, since they do things like magically taking a raw memory image and automatically providing the plugin with an appropriate Intel translation layer and an accurate symbol table without either the plugin or the calling program having to specify all the necessary details.

Note: Volatility 2 used to do this as well, but it wasn't a particularly modular mechanism, and was used only for stacking address spaces (rather than identifying profiles), and it couldn't really be disabled/configured easily. Automagics in Volatility 3 are a core component which consumers of the library can call or not at their discretion.

WRITING PLUGINS

2.1 How to Write a Simple Plugin

This guide will step through how to construct a simple plugin using Volatility 3.

The example plugin we'll use is *DllList*, which features the main traits of a normal plugin, and reuses other plugins appropriately.

Note: This document will not include the complete code necessary for a working plugin (such as imports, etc) since it's designed to focus on the necessary components for writing a plugin. For complete and functioning plugins, the `framework/plugins` directory should be consulted.

2.1.1 Inherit from PluginInterface

The first step is to define a class that inherits from *PluginInterface*. Volatility automatically finds all plugins defined under the various plugin directories by importing them and then making use of any classes that inherit from *PluginInterface*.

```
from volatility3.framework import interfaces

class DllList(interfaces.plugins.PluginInterface):
```

The next step is to define the requirements of the plugin, these will be converted into options the user can provide based on the User Interface.

2.1.2 Define the plugin requirements

These requirements are the names of variables that will need to be populated in the configuration tree for the plugin to be able to run properly. Any that are defined as optional need not necessarily be provided.

```
_version = (1, 0, 0)
_required_framework_version = (2, 0, 0)

@classmethod
def get_requirements(cls):
    return [requirements.ModuleRequirement(name = 'kernel', description = 'Windows kernel
↪',
                                           architectures = ["Intel32", "Intel64"])]
```

(continues on next page)

(continued from previous page)

```

requirements.ListRequirement(name = 'pid',
                             element_type = int,
                             description = "Process IDs to include (all
↳ other processes are excluded)",
                             optional = True),
requirements.PluginRequirement(name = 'pslist',
                               plugin = pslist.PsList,
                               version = (2, 0, 0))]
```

This is a classmethod, because it is called before the specific plugin object has been instantiated (in order to know how to instantiate the plugin). At the moment these requirements are fairly straightforward:

```

requirements.ModuleRequirement(name = 'kernel', description = 'Windows kernel',
                              architectures = ["Intel32", "Intel64"]),
```

This requirement specifies the need for a particular submodule. Each module requires a *TranslationLayer* and a *SymbolTable*, which are fulfilled by two subrequirements: a *TranslationLayerRequirement* and a *SymbolTableRequirement*. At the moment, the automagic only fills *ModuleRequirements* with kernels, and so has relatively few parameters. It requires the architecture for the underlying *TranslationLayer*, and the offset of the module within that layer.

The name of the module will be stored in the `kernel` configuration option, and the module object itself can be accessed from the `context.modules` collection. This requirement is a Complex Requirement and therefore will not be requested directly from the user.

Note: In previous versions of volatility 3, there was no *ModuleRequirement*, and instead two requirements were defined a *TranslationLayer* and a *SymbolTableRequirement*. These still exist, and can be used, most plugins just define a single *ModuleRequirement* for the kernel, which the automagic will populate. The *ModuleRequirement* has two automatic sub-requirements, a *TranslationLayerRequirement* and a *SymbolTableRequirement*, but the module also includes the offset of the module, and will allow future expansion to specify specific modules when application level plugins become more common. Below are how the requirements would be specified:

```

requirements.TranslationLayerRequirement(name = 'primary',
                                         description = 'Memory layer for the kernel',
                                         architectures = ["Intel32", "Intel64"]),
```

This requirement indicates that the plugin will operate on a single *TranslationLayer*. The name of the loaded layer will appear in the plugin's configuration under the name `primary`. Requirement values can be accessed within the plugin through the plugin's *config* attribute (for example `self.config['pid']`).

Note: The name itself is dynamic depending on the other layers already present in the Context. Always use the value from the configuration rather than attempting to guess what the layer will be called.

Finally, this defines that the translation layer must be on the Intel Architecture. At the moment, this acts as a filter, failing to be satisfied by memory images that do not match the architecture required.

Most plugins will only operate on a single layer, but it is entirely possible for a plugin to request two different layers, for example a plugin that carries out some form of difference or statistics against multiple memory images.

This requirement (and the next two) are known as Complex Requirements, and user interfaces will likely not directly request a value for this from a user. The value stored in the configuration tree for a *TranslationLayerRequirement* is the string name of a layer present in the context's memory that satisfies the requirement.

```
requirements.SymbolTableRequirement(name = "nt_symbols",
                                     description = "Windows kernel symbols"),
```

This requirement specifies the need for a particular *SymbolTable* to be loaded. This gets populated by various Automagic as the nearest sibling to a particular *TranslationLayerRequirement*. This means that if the *TranslationLayerRequirement* is satisfied and the Automagic can determine the appropriate *SymbolTable*, the name of the *SymbolTable* will be stored in the configuration.

This requirement is also a Complex Requirement and therefore will not be requested directly from the user.

```
requirements.ListRequirement(name = 'pid',
                             description = 'Filter on specific process IDs',
                             element_type = int,
                             optional = True),
```

The next requirement is a List Requirement, populated by integers. The description will be presented to the user to describe what the value represents. The optional flag indicates that the plugin can function without the pid value being defined within the configuration tree at all.

```
requirements.PluginRequirement(name = 'pslist',
                               plugin = pslist.PsList,
                               version = (2, 0, 0))]
```

This requirement indicates that the plugin will make use of another plugin's code, and specifies the version requirements on that plugin. The version is specified in terms of Semantic Versioning meaning that, to be compatible, the major versions must be identical and the minor version must be equal to or higher than the one provided. This requirement does not make use of any data from the configuration, even if it were provided, it is merely a functional check before running the plugin. To define the version of a plugin, populate the *_version* class variable as a tuple of version numbers (*major, minor, patch*). So for example:

```
_version = (1, 0, 0)
```

The plugin may also require a specific version of the framework, and this also uses Semantic Versioning, and can be set by defining the *_required_framework_version*. The major version should match the version of volatility the plugin is to be used with, which at the time of writing would be 2.2.0, and so would be specified as below. If only features, for example, from 2.0.0 are used, then the lowest applicable version number should be used to support the greatest number of installations:

```
_required_framework_version = (2, 0, 0)
```

2.1.3 Define the *run* method

The run method is the primary method called on a plugin. It takes no parameters (these have been passed through the context's configuration tree, and the context is provided at plugin initialization time) and returns an unpopulated *TreeGrid* object. These are typically constructed based on a generator that carries out the bulk of the plugin's processing. The *TreeGrid* also specifies the column names and types that will be output as part of the *TreeGrid*.

```
def run(self):

    filter_func = pslist.PsList.create_pid_filter(self.config.get('pid', None))
    kernel = self.context.modules[self.config['kernel']]
```

(continues on next page)

(continued from previous page)

```

return renderers.TreeGrid([("PID", int),
                           ("Process", str),
                           ("Base", format_hints.Hex),
                           ("Size", format_hints.Hex),
                           ("Name", str),
                           ("Path", str)],
                           self._generator(pslist.PsList.list_processes(self.context,
                                                                       kernel.layer_
↪name,
                                                                       kernel.symbol_
↪table_name,
                                                                       filter_func =
↪filter_func)))

```

In this instance, the plugin constructs a filter (using the PsList plugin's *classmethod* for creating filters). It checks the plugin's configuration for the pid value, and passes it in as a list if it finds it, or None if it does not. The *create_pid_filter()* method accepts a list of process identifiers that are included in the list. If the list is empty, all processes are returned.

The next line specifies the columns by their name and type. The types are simple types (int, str, bytes, float, and bool) but can also provide hints as to how the output should be displayed (such as a hexadecimal number, using *volatility3.framework.renderers.format_hints.Hex*). This indicates to user interfaces that the value should be displayed in a particular way, but does not guarantee that the value will be displayed that way (for example, if it doesn't make sense to do so in a particular interface).

Finally, the generator is provided. The generator accepts a list of processes, which is gathered using a different plugin, the *PsList* plugin. That plugin features a *classmethod*, so that other plugins can call it. As such, it takes all the necessary parameters rather than accessing them from a configuration. Since it must be portable code, it takes a context, as well as the layer name, symbol table and optionally a filter. In this instance we unconditionally pass it the values from the configuration for the layer and symbol table from the kernel module object, constructed from the kernel configuration requirement. This will generate a list of *EPROCESS* objects, as provided by the PsList plugin, and is not covered here but is used as an example for how to share code across plugins (both as the provider and the consumer of the shared code).

2.1.4 Define the generator

The *TreeGrid* can be populated without a generator, but it is quite a common model to use. This is where the main processing for this plugin lives.

```

def _generator(self, procs):

    for proc in procs:

        for entry in proc.load_order_modules():

            BaseDllName = FullDllName = renderers.UnreadableValue()
            try:
                BaseDllName = entry.BaseDllName.get_string()
                # We assume that if the BaseDllName points to an invalid buffer, so will
↪FullDllName
                FullDllName = entry.FullDllName.get_string()
            except exceptions.InvalidAddressException:

```

(continues on next page)

(continued from previous page)

```

        pass

        yield (0, (proc.UniqueProcessId,
                    proc.ImageFileName.cast("string", max_length = proc.ImageFileName.
↪ vol.count,
                                                errors = 'replace'),
                    format_hints.Hex(entry.DllBase), format_hints.Hex(entry.
↪ SizeOfImage),
                    BaseDllName, FullDllName))

```

This iterates through the list of processes and for each one calls the `load_order_modules()` method on it. This provides a list of the loaded modules within the process.

The plugin then defaults the `BaseDllName` and `FullDllName` variables to an `UnreadableValue`, which is a way of indicating to the user interface that the value couldn't be read for some reason (but that it isn't fatal). There are currently four different reasons a value may be unreadable:

- **Unreadable**: values which are empty because the data cannot be read
- **Unparsable**: values which are empty because the data cannot be interpreted correctly
- **NotApplicable**: values which are empty because they don't make sense for this particular entry
- **NotAvailable**: values which cannot be provided now (but might in a future run, via new symbols or an updated plugin)

This is a safety provision to ensure that the data returned by the Volatility library is accurate and describes why information may not be provided.

The plugin then takes the process's `BaseDllName` value, and calls `get_string()` on it. All structure attributes, as defined by the symbols, are directly accessible and use the case-style of the symbol library it came from (in Windows, attributes are CamelCase), such as `entry.BaseDllName` in this instance. Any attributes not defined by the symbol but added by Volatility extensions cannot be properties (in case they overlap with the attributes defined in the symbol libraries) and are therefore always methods and prepended with `get_`, in this example `BaseDllName.get_string()`.

Finally, `FullDllName` is populated. These operations read from memory, and as such, the memory image may be unable to read the data at a particular offset. This will cause an exception to be thrown. In Volatility 3, exceptions are thrown as a means of communicating when something exceptional happens. It is the responsibility of the plugin developer to appropriately catch and handle any non-fatal exceptions and otherwise allow the exception to be thrown by the user interface.

In this instance, the `InvalidAddressException` class is caught, which is thrown by any layer which cannot access an offset requested of it. Since we have already populated both values with `UnreadableValue` we do not need to write code for the exception handler.

Finally, we yield the record in the format required by the `TreeGrid`, a tuple, listing the indentation level (for trees) and then the list of values for each column. This plugin demonstrates casting a value `ImageFileName` to ensure it's returned as a string with a specific maximum length, rather than its original type (potentially an array of characters, etc). This is carried out using the `cast()` method which takes a type (either a native type, such as string or pointer, or a structure type defined in a `SymbolTable` such as `<table>!_UNICODE`) and the parameters to that type.

Since the cast value must populate a string typed column, it had to be a Python string (such as being cast to the native type string) and could not have been a special Structure such as `_UNICODE`. For the format hint columns, the format hint type must be used to ensure the error checking does not fail.

2.2 Writing more advanced Plugins

There are several common tasks you might wish to accomplish, there is a recommended means of achieving most of these which are discussed below.

2.2.1 Writing Reusable Methods

Classes which inherit from `PluginInterface` all have a `run()` method which takes no parameters and will return a `TreeGrid`. Since most useful functions are parameterized, to provide parameters to a plugin the *configuration* for the context must be appropriately manipulated. There is scope for this, in order to run multiple plugins (see *Writing plugins that run other plugins*) but a much simpler method is to provide a parameterized *classmethod* within the plugin, which will allow the method to yield whatever kind of output it will generate and take whatever parameters it might need.

This is how processes are listed, which is an often used function. The code lives within the `PsList` plugin but can be used by other plugins by providing the appropriate parameters (see `list_processes()`). It is up to the author of a plugin to validate that any required plugins are present and are the appropriate version.

2.2.2 Writing plugins that run other plugins

Occasionally plugins will want to process the output from other plugins (for example, the *timeliner* plugin which runs all other available plugins that feature a *Timeliner* interface). This can be achieved with the following example code:

```
automagics = automagic.choose_automagic(automagic.available(self._context), plugin_class)
plugin = plugins.construct_plugin(self.context, automagics, plugin_class, self.config_
↳ path,
                                self._progress_callback, self.open)
```

This code will first generate suitable automagics for running against the context. Unfortunately this must be re-run for each plugin in order to populate the context's configuration correctly based on the plugin's requirements (which may vary between plugins). Once the automagics have been constructed, the plugin can be instantiated using the helper function `construct_plugin()` providing:

- the base context (containing the configuration and any already loaded layers or symbol tables),
- the plugin class to run,
- the configuration path within the context for the plugin
- any callback to determine progress in lengthy operations
- an open method for the plugin to create files during the run

With the constructed plugin, it can either be run by calling its `run()` method, or any other known method can be invoked on it.

2.2.3 Writing plugins that output files

Every plugin can create files, but since the user interface must decide how to actually provide these files to the user, an abstraction layer is used.

The user interface specifies an `open_method` (which is actually a class constructor that can double as a python `ContextManager`, so it can be used by the python `with` keyword). This is set on the plugin using `plugin.set_open_method` and can then be called or accessed using `plugin.open(preferred_filename)`. There are no additional options that can be set on the filename, and a `FileHandlerInterface` is the result. This mimics an `IO[bytes]` object, which closely mimics a standard python file-like object.

As such code for outputting to a file would be expected to look something like:

```
with self.open(preferred_filename) as file_handle:
    file_handle.write(data)
```

Since `self.open` returns a `ContextManager` the file is closed automatically and thus committed for the UI to process as necessary. If the file is not closed, the UI may not be able to properly process it and unexpected results may arise. In certain instances you may receive a `file_handle` from another plugin's method, in which case the file is unlikely to be closed to allow the preferred filename to be changed (or data to be added/modified, if necessary).

2.2.4 Writing Scanners

Scanners are objects that adhere to the `ScannerInterface`. They are passed to the `scan()` method on layers which will divide the provided range of sections (or the entire layer if none are provided) and call the `ScannerInterface()`'s `call_method` method with each chunk as a parameter, ensuring a suitable amount of overlap (as defined by the scanner). The offset of the chunk, within the layer, is also provided as a parameter.

Scanners can technically maintain state, but it is not recommended since the ordering that the chunks are scanned is not guaranteed. Scanners may be executed in parallel if they mark themselves as `thread_safe` although the threading technique may be either standard threading or multiprocessing. Note, the only component of the scans which is parallelized are those that go on within the scan method. As such, any processing carried out on the results yielded by the scanner will be processed in serial. It should also be noted that generating the addresses to be scanned are not iterated in parallel (in full, before the scanning occurs), meaning the smaller the sections to scan the quicker the scan will run.

Empirically it was found that scanners are typically not the most time intensive part of plugins (even those that do extensive scanning) and so parallelism does not offer significant gains. As such, parallelism is not enabled by default but interfaces can easily enable parallelism when desired.

2.2.5 Writing/Using Intermediate Symbol Format Files

It can occasionally be useful to create a data file containing the static structures that can create a `Template` to be instantiated on a layer. Volatility has all the machinery necessary to construct these for you from properly formatted JSON data.

The JSON format is documented by the JSON schema files located in schemas. These are versioned using standard .so library versioning, so they may not increment as expected. Each schema lists an available version that can be used, which specifies five different sections:

- `Base_types` - These are the basic type names that will make up the native/primitive types
- `User_types` - These are the standard definitions of type structures, most will go here
- `Symbols` - These list offsets that are associated with specific names (and can be associated with specific type names)
- `Enums` - Enumerations that offer a number of choices

- Metadata - This is information about the generator, when the file was generated and similar

Constructing an appropriate file, the file can be loaded into a symbol table as follows:

```
table_name = intermed.IntermediateSymbolTable.create(context, config_path, 'sub_path',
↪ 'filename')
```

This code will load a JSON file from one of the standard symbol paths (volatility3/symbols and volatility3/framework/symbols) under the additional directory `sub_path`, with a name matching `filename.json` (the extension should not be included in the filename).

The `sub_path` parameter acts as a filter, so that similarly named symbol tables for each operating system can be addressed separately. The top level directories which `sub_path` filters are also checked as zipfiles to determine any symbols within them. As such, group of symbol tables can be included in a single zip file. The filename for the symbol tables should not contain an extension, as extensions for JSON (and compressed JSON files) will be tested to find a match.

Additional parameters exist, such as `native_types` which can be used to provide pre-populated native types.

Another useful parameter is `table_mapping` which allows for type referenced inside the JSON (such as `one_table!type_name`) would allow remapping of `one_table` to `another_table` by providing a dictionary as follows:

```
table_name = intermed.IntermediateSymbolTable.create(context, config_path, 'sub_path',
↪ 'filename',
    table_mapping = {'one_table': 'another_table'})
```

The last parameter that can be used is called `class_types` which allows a particular structure to be instantiated on a class other than `StructType`, allowing for additional methods to be defined and associated with the type.

The table name can then be used to access the constructed table from the context, such as:

```
context.symbol_space[table_name]
```

2.2.6 Writing new Translation Layers

Translation layers offer a way for data to be translated from a higher (domain) layer to a lower (range) layer. The main method that must be overloaded for a translation layer is the `mapping` method. Usually this is a linear mapping whereby a value at an offset in the domain maps directly to an offset in the range.

Most new layers should inherit from `LinearlyMappedLayer` where they can define a mapping method as follows:

```
def mapping(self,
            offset: int,
            length: int,
            ignore_errors: bool = False) -> Iterable[Tuple[int, int, int, int, str]]:
```

This takes a (domain) offset and a length of block, and returns a sorted list of chunks that cover the requested amount of data. Each chunk contains the following information, in order:

offset (domain offset)

requested offset in the domain

chunk length

the length of the data in the domain

mapped offset (range offset)

where the data lives in the lower layer

mapped length

the length of the data in the range

layer_name

the layer that this data comes from

An example (and the most common layer encountered in memory forensics) would be an Intel layer, which models the intel page mapping system. Based on a series of tables stored within the layer itself, an intel layer can convert a virtual address to a physical address. It should be noted that intel layers allow multiple virtual addresses to map to the same physical address (but a single virtual address cannot ever map to more than one physical address).

As a simple example, in a virtual layer which looks like *abracadabra* but maps to a physical layer that looks like *abcd*, requesting *mapping(5, 4)* would return:

```
[(5,1,0,1, 'physical_layer'),
 (6,1,3,1, 'physical_layer'),
 (7,2,0,2, 'physical_layer')
]
```

This mapping mechanism allows for great flexibility in that chunks making up a virtual layer can come from multiple different range layers, allowing for swap space to be used to construct the virtual layer, for example. Also, by defining the mapping method, the read and write methods (which read and write into the domain layer) are defined for you to write to the lower layers (which in turn can write to layers even lower than that) until eventually they arrive at a *DataLayer*, such as a file or a buffer.

This mechanism also allowed for some minor optimization in scanning such a layer, but should further control over the scanning of layers be needed, please refer to the Layer Scanning page.

Whilst it may seem as though some of the data seems redundant (the length values are always the same) this is not the case for *NonLinearlySegmentedLayer*. These layers do not guarantee that each domain address maps directly to a range address, and in fact can carry out processing on the data. These layers are most commonly encountered as compression or encryption layers (whereby a domain address may map into a chunk of the range, but not directly). In this instance, the mapping will likely define additional methods that can take a chunk and process it from its original value into its final value (such as decompressing for read and compressing for write).

These methods are private to the class, and are used within the standard *read* and *write* methods of a layer. A non-linear layer's mapping method should return the data required to be able to return the original data. As an example, a run length encoded layer, whose domain data looks like *aaabbbbbcdddd* could be stored as *3a5b1c4d*. The mapping method call for *mapping(5,4)* should return all the regions that encompass the data required. The layer would return the following data:

```
[(5, 4, 2, 4, 'rle layer')]
```

It would then define *_decode* and *_encode* methods that could convert from one to the other. In the case of *read(5, 4)*, the *_decode* method would be provided with the following parameters:

```
data = "5b1c"
mapped_offset = 2
offset = 5
output_length = 4
```

This requires that the *_decode* method can unpack the encoding back to *bbbbc* and also know that the decoded block starts at 3, so that it can return just *bbbc*, as required. Such layers therefore typically need to keep much more internal state, to keep track of which offset of encoded data relates to which decoded offset for both the mapping and *_encode* and *_decode* methods.

If the data processing produces known fixed length values, then it is possible to write an *_encode* method in much the same way as the decode method. *_encode* is provided with the data to encode, the mapped_offset to write it to the lower (range) layer, the original offset of the data in the higher (domain) layer and the value of the not yet encoded data to write. The encoded result, regardless of length will be written over the current image at the mapped_offset. No other changes or updates to tables, etc are carried out.

`_encode` is much more difficult if the encoded data can be variable length, as it may involve rewriting most, if not all of the data in the image. Such a situation is not currently supported with this API and it is strongly recommended to raise `NotImplementedError` in this method.

Communicating between layers

Layers can ask for information from lower layers using the `layer.metadata` lookup. In the following example, a `LayerStacker` automagic that generates the intel `TranslationLayer` requests whether the base layer knows what the `page_map_offset` value should be, a `CrashDumpLayer` would have that information. As such the `TranslationLayer` would just lookup the `page_map_offset` value in the `base_layer.metadata` dictionary:

```
if base_layer.metadata.get('page_layer_offset', None) is not None:
```

Most layers will return `None`, since this is the default, but the `CrashDumpLayer` may know what the value should be, so it therefore populates the `metadata` property. This is defined as a read-only mapping to ensure that every layer includes data from every underlying layer. As such, `CrashDumpLayer` would actually specify this value by setting it in the protected dictionary by `self._direct_metadata['page_map_offset']`.

There is, unfortunately, no easy way to form consensus between a particular layer may want and what a particular layer may be able to provide. At the moment, the main information that layers may populate are:

- `os` with values of *Windows*, *Linux*, *Mac* or *unknown*
- `architecture` with values of *Intel32*, *Intel64* or *unknown*
- `pae` a boolean specifying whether the PAE mode is enabled for windows
- `page_map_offset` the value pointing to the intel `page_map_offset`

Any value can be specified and used by layers but consideration towards ambiguity should be used to ensure that overly generic names aren't used for something and then best describe something else that may be needed later on.

Note: The data stored in metadata is *not* restored when constructed from a configuration, so metadata should only be used as a temporary means of storing information to be used in constructing later objects and all information required to recreate an object must be written through the requirements mechanism.

2.2.7 Writing new Templates and Objects

In most cases, a whole new type of object is unnecessary. It will usually be derived from an `StructType` (which is itself just another name for a `AggregateType`, but it's better to use `StructType` for readability).

This can be used as a class override for a particular symbol table, so that an existing structure can be augmented with additional methods. An example of this would be:

```
symbol_table = contexts.symbol_space[symbol_table_name]
symbol_table.set_type_class('<structure_name>', NewStructureClass)
```

This will mean that when a specific structure is loaded from the `symbol_space`, it is not constructed as a standard `StructType`, but instead is instantiated using the `NewStructureClass`, meaning new methods can be called directly on it.

If the situation really calls for an entirely new object, that isn't covered by one of the existing `PrimitiveObject` objects (such as `Integer`, `Boolean`, `Float`, `Char`, `Bytes`) or the other builtins (such as `Array`, `Bitfield`, `Enumeration`, `Pointer`, `String`, `Void`) then you can review the following information about defining an entirely new object.

All objects must inherit from *ObjectInterface* which defines a constructor that takes a context, a *type_name*, an *ObjectInformation* object and then can accept additional keywords (which will not necessarily be provided if the object is constructed from a JSON reference).

The *ObjectInformation* class contains all the basic elements that define an object, which include:

- *layer_name*
- *offset*
- *member_name*
- *parent*
- *native_layer_name*
- *size*

The *layer_name* and *offset* are how volatility reads the data of the object. Since objects can reference other objects (specifically pointers), and contain values that are used as offsets in a particular layer, there is also the concept of a *native_layer_name*. The *native_layer_name* allows an object to be constructed based on physical data (for instance) but to reference virtual addresses, or for an object in the kernel virtual layer to reference offsets in a process virtual layer.

The *member_name* and *parent* are optional and are used for when an object is constructed as a member of a structure. The *parent* points back to the object that created this one, and *member_name* is the name of the attribute of the parent used to get to this object.

Finally, some objects are dynamically sized, and this *size* parameter allows a constructor to specify how big the object should be. Note, the *size* can change throughout the lifespan of the object, and the object will need to ensure that it compensates for such a change.

Objects must also contain a specific class called *VolTemplateProxy* which must inherit from *ObjectInterface*. This is used to access information about a structure before it has been associated with data and becomes an Object. The *VolTemplateProxy* class contains a number of abstract classmethods, which take a *Template*. The main method that is likely to need overwriting is the *size* method, which should return the size of the object (for the template of a dynamically-sized object, this should be a suitable value, and calculated based on the best available information). For most objects, this can be determined from the JSON data used to construct a normal *Struct* and therefore only needs to be defined for very specific objects.

2.3 Using Volatility 3 as a Library

This portion of the documentation discusses how to access the Volatility 3 framework from an external application.

The general process of using volatility as a library is to as follows:

1. *Creating a context*
2. (Optional) *Determine what plugins are available*
3. (Optional) *Determine what configuration options a plugin requires*
4. *Set the configuration in the context*
5. (Optional) *Using automagic to complete the configuration*
6. *Run the plugin*
7. *Render the TreeGrid*

2.3.1 Creating a context

First we make sure the volatility framework works the way we expect it (and is the version we expect). The versioning used is semantic versioning, meaning any version with the same major number and a higher or equal minor number will satisfy the requirement. An example is below since the CLI doesn't need any of the features from versions 1.1 or 1.2:

```
volatility3.framework.require_interface_version(1, 0, 0)
```

Contexts can be spun up quite easily, just construct one. It's not a singleton, so multiple contexts can be constructed and operate independently, but be aware of which context you're handing where and make sure to use the correct one. Typically once a context has been handed to a plugin, all objects will be created with a reference to that context.

```
ctx = contexts.Context() # Construct a blank context
```

2.3.2 Determine what plugins are available

You can also interrogate the framework to see which plugins are available. First we have to try to load all available plugins. The `import_files()` method will automatically use the module paths for the provided module (in this case, `volatility3.plugins`) and walk the directory (or directories) loading up all python files. Any import failures will be provided in the failures return value, unless the second parameter is `False` in which case the call will raise any exceptions encountered. Any additional directories containing plugins should be added to the `__path__` attribute for the `volatility3.plugins` module. The standard paths should generally also be included, which can be found in `volatility3.constants.PLUGINS_PATH`.

```
volatility3.plugins.__path__ = <new_plugin_path> + constants.PLUGINS_PATH
failures = framework.import_files(volatility3.plugins, True)
```

Note: Volatility uses the `volatility3.plugins` namespace for all plugins (including those in `volatility3.framework.plugins`). Please ensure you only use `volatility3.plugins` and only ever import plugins from this namespace. This ensures the ability of users to override core plugins without needing write access to the framework directory.

Once the plugins have been imported, we can interrogate which plugins are available. The `list_plugins()` call will return a dictionary of plugin names and the plugin classes.

```
plugin_list = framework.list_plugins()
```

2.3.3 Determine what configuration options a plugin requires

For each plugin class, we can call the classmethod `get_requirements()` on it, which will return a list of objects that adhere to the `RequirementInterface` method. The various types of Requirement are split roughly in two, `SimpleTypeRequirement` (such as integers, booleans, floats and strings) and more complex requirements (such as lists, choices, multiple requirements, translation layer requirements or symbol table requirements). A requirement just specifies a type of data and a name, and must be combined with a configuration hierarchy to have meaning.

List requirements are a list of simple types (integers, booleans, floats and strings), choices must match the available options, multiple requirements needs all their subrequirements fulfilled and the other types require the names of valid translation layers or symbol tables within the context, respectively. Luckily, each of these requirements can tell you whether they've been fulfilled or not later in the process. For now, they can be used to ask the user to fill in any parameters they made need to. Some requirements are optional, others are not.

The plugin is essentially a multiple requirement. It should also be noted that automagic classes can have requirements (as can translation layers).

2.3.4 Set the configuration in the context

Once you know what requirements the plugin will need, you can populate them within the *context.config*. The configuration is essentially a hierarchical tree of values, much like the windows registry. Each plugin is instantiated at a particular branch within the hierarchy and will look for its configuration options under that hierarchy (if it holds any configurable items, it will likely instantiate those at a point underneath its own branch). To set the hierarchy, you'll need to know where the configurables will be constructed.

For this example, we'll assume plugins' *base_config_path* is set as *plugins*, and that automagics are configured under the *automagic* tree. We'll see later how to ensure this matches up with the plugins and automagic when they're constructed. Joining configuration options should always be carried out using *path_join()* in case the separator value gets changed in the future. Configuration items can then be set as follows:

```
config_path = path_join(base_config_path, plugin.__class__.__name__, <plugin_parameter>)
context.config['plugins.<plugin_class_name>.<plugin_parameter>'] = value
```

2.3.5 Using automagic to complete the configuration

Many of the options will require a lot of construction (layers on layers on layers). The automagic functionality is there to help take some of that burden away. There are automagics designed to stack layers (such as compression and file formats, as well as architectures) and automagics for determining critical information from windows, linux and mac layers about the operating system. The list of available automagics can be found using:

```
available_automagics = automagic.available(ctx)
```

This again, will require that all automagic modules have been loaded but this should happen simply as part of importing the *automagic* module. The available list will be pre-instantiated copies of the automagic with their configuration path and context provided (based on *constants.AUTOMAGIC_CONFIG_PATH* and the automagic class name).

A suitable list of automagics for a particular plugin (based on operating system) can be found using:

```
automagics = automagic.choose_automagic(available_automagics, plugin)
```

This will take the plugin module, extract the operating system (first level of the hierarchy) and then return just the automagics which apply to the operating system. Each automagic can exclude itself from being used for specific operating systems, so that an automagic designed for linux is not used for windows or mac plugins.

These automagics can then be run by providing the list, the context, the plugin to be run, the hierarchy name that the plugin will be constructed on ('plugins' by default) and a *progress_callback*. This is a callable which takes a percentage of completion and a description string and will be called throughout the process to indicate to the user how much progress has been made.

```
errors = automagic.run(automagics, context, plugin, base_config_path, progress_callback,
↳ progress_callback)
```

Any exceptions that occur during the execution of the automagic will be returned as a list of exceptions.

2.3.6 Run the plugin

Firstly, we should check whether the plugin will be able to run (ie, whether the configuration options it needs have been successfully set). We do this as follow (where `plugin_config_path` is the `base_config_path` (which defaults to *plugins* and then the name of the class itself):

```
unsatisfied = plugin.unsatisfied(context, plugin_config_path)
```

If `unsatisfied` is an empty list, then the plugin has been given everything it requires. If not, it will be a Dictionary of the hierarchy paths and their associated requirements that weren't satisfied.

The plugin can then be instantiated with the context (containing the plugin's configuration) and the path that the plugin can find its configuration at. This configuration path only needs to be a unique value to identify where the configuration details can be found, similar to a registry key in Windows.

A `progress_callback` can also be provided to give users feedback whilst the plugin is running. A progress callback is a function (callable) that takes a percentage and a descriptive string. User interfaces implementing these can therefore provide progress feedback to a user, as the framework will call these every so often during intensive actions, to update the user as to how much has been completed so far.

Also, should the plugin produce files, an `open_method` can be set on the plugin, which will be called whenever a plugin produces an auxiliary file.

```
constructed = plugin(context, plugin_config_path, progress_callback = progress_callback)
constructed.set_open_method(file_handler)
```

The `file_handler` must adhere to the [FileHandlerInterface](#), which represents an `IO[bytes]` object but also contains a `preferred_filename` attribute as a hint indicating what the file being produced should be called. When a plugin produces a new file, rather than opening it with the python `open` method, it will use the `FileHandlerInterface` and construct it with a descriptive filename, and then write bytes to it using the `write` method, just like other python file-like objects. This allows web user interfaces to offer the files for download, whilst CLIs to write them to disk and other UIs to handle files however they need.

All of this functionality has been condensed into a framework method called `construct_plugin` which will take and run the automagics, and instantiate the plugin on the provided `base_config_path`. It also accepts an optional `progress_callback` and an optional `file_consumer`.

```
constructed = plugins.construct_plugin(ctx, automagics, plugin, base_config_path,
↪progress_callback, file_consumer)
```

Finally the plugin can be run, and will return a [TreeGrid](#).

```
treegrid = constructed.run()
```

2.3.7 Render the TreeGrid

The results are now in a structure of rows, with a hierarchy (allowing a row to be a child of another row).

The `TreeGrid` can tell you what columns it contains, and the types of each column, but does not contain any data yet. It must first be populated. This actually iterates through the results of the plugin, which may have been provided as a generator, meaning this step may take the actual processing time, whilst the plugin does the actual work. This can return an exception if one occurs during the running of the plugin.

The results can be accessed either as the results are being processed, or by visiting the nodes in the tree once it is fully populated. In either case, a visitor method will be required. The visitor method should accept a [TreeNode](#) and an *accumulator*. It will return an updated accumulator.

When provided a [TreeNode](#), it can be accessed as a dictionary based on the column names that the treegrid contains. It should be noted that each column can contain only the type specified in the *column.type* field (which can be a simple type like string, integer, float, bytes or a more complex type, like a `DateTime`, a `Disassembly` or a descendant of [BaseAbsentValue](#)). The various fields may also be wrapped in *format_hints* designed to tell the user interface how to render the data. These hints can be things like `Bin`, `Hex` or `HexBytes`, so that fields like offsets are displayed in hex form or so that bytes are displayed in their hex form rather than their raw form. Descendants of [BaseAbsentValue](#) can currently be one of [UnreadableValue](#), [UnparsableValue](#), [NotApplicableValue](#) or [NotAvailableValue](#). These indicate that data could not be read from the memory for some reason, could not be parsed properly, was not applicable or was not available.

A simple text renderer (that returns output immediately) would appear as follows. This doesn't use the accumulator, but instead uses `print` to directly produce the output. This is not recommended:

```
for column in grid.columns:
    print(column.name)

def visitor(node, _accumulator):
    # Nodes always have a path value, giving them a path_depth of at least 1, we use max_
    ↪ just in case
    print(" " * max(0, node.path_depth - 1), end = " ")
    for column_index in range(len(grid.columns)):
        column = grid.columns[column_index]
        print(repr(node.values[column_index]), end = '\t')

    print('')
    return None

grid.populate(visitor, None)
```

More complex examples of renderers can be found in the default CLI implementation, such as the [QuickTextRenderer](#) or the [PrettyTextRenderer](#).

CREATING NEW SYMBOL TABLES

This page details how symbol tables are located and used by Volatility, and documents the tools and methods that can be used to make new symbol tables.

3.1 How Volatility finds symbol tables

All files are stored as JSON data, they can be in pure JSON files as `.json`, or compressed as `.json.gz` or `.json.xz`. Volatility will automatically decompress them on use. It will also cache their contents (compressed) when used, located under the user's home directory, in `.cache/volatility3`, along with other useful data. The cache directory currently cannot be altered.

Symbol table JSON files live, by default, under the `volatility3/symbols` directory. The symbols directory is configurable within the framework and can usually be set within the user interface.

These files can also be compressed into ZIP files, which Volatility will process in order to locate symbol files.

Volatility maintains a cache mapping the appropriate identifier for each symbol file against its filename. This cache is updated by automagic called as part of the standard automagic that's run each time a plugin is run. If a large number of new symbols file are detected, this may take some time, but can be safely interrupted and restarted and will not need to run again as long as the symbol files stay in the same location.

3.2 Windows symbol tables

For Windows systems, Volatility accepts a string made up of the GUID and Age of the required PDB file. It then searches all files under the configured symbol directories under the windows subdirectory. Any that contain metadata which matches the pdb name and GUID/age (or any compressed variant) will be used. If such a symbol table cannot be found, then the associated PDB file will be downloaded from Microsoft's Symbol Server and converted into the appropriate JSON format, and will be saved in the correct location.

Windows symbol tables can be manually constructed from an appropriate PDB file. The primary tool for doing this is built into Volatility 3, called `pdbconv.py`. It can be run from the top-level Volatility path, using the following command:

```
PYTHONPATH="." python volatility3/framework/symbols/windows/pdbconv.py
```

The `PYTHONPATH` environment variable is not required if the Volatility library is installed in the system's library path or a virtual environment.

3.3 Mac or Linux symbol tables

For Mac/Linux systems, both use the same mechanism for identification. The generated files contain an identifying string (the operating system banner), which Volatility's automagic can detect. Volatility caches the mapping between the strings and the symbol tables they come from, meaning the precise file names don't matter and can be organized under any necessary hierarchy under the symbols directory.

Linux and Mac symbol tables can be generated from a DWARF file using a tool called [dwarf2json](#). Currently a kernel with debugging symbols is the only suitable means for recovering all the information required by most Volatility plugins. Note that in most linux distributions, the standard kernel is stripped of debugging information and the kernel with debugging information is stored in a package that must be acquired separately.

A generic table isn't guaranteed to produce accurate results, and would reduce the number of structures that all plugins could rely on. As such, and because linux kernels with different configurations can produce different structures, volatility 3 requires that the banners in the JSON file match the banners found in the image *exactly*, not just the version number. This can include elements such as the compilation time and even the version of gcc used for the compilation. The exact match is required to ensure that the results volatility returns are accurate, therefore there is no simple means provided to get the wrong JSON ISF file to easily match.

To determine the string for a particular memory image, use the *banners* plugin. Once the specific banner is known, try to locate that exact kernel debugging package for the operating system. Unfortunately each distribution provides its debugging packages under different package names and there are so many that the distribution may not keep all old versions of the debugging symbols, and therefore **it may not be possible to find the right symbols to analyze a linux memory image with volatility**. With Macs there are far fewer kernels and only one distribution, making it easier to ensure that the right symbols can be found.

Once a kernel with debugging symbols/appropriate DWARF file has been located, [dwarf2json](#) will convert it into an appropriate JSON file. Example code for automatically creating a JSON from URLs for the kernel debugging package and the package containing the System.map, can be found in [stock-linux-json.py](#). The System.map file is recommended for completeness, but a kernel with debugging information often contains the same symbol offsets within the DWARF data, which dwarf2json can extract into the JSON ISF file.

The banners available for volatility to use can be found using the *isinfo* plugin, but this will potentially take a long time to run depending on the number of JSON files available. This will list all the JSON (ISF) files that volatility3 is aware of, and for linux/mac systems what banner string they search for. For volatility to use the JSON file, the banners must match exactly (down to the compilation date).

Note: Steps for constructing a new kernel ISF JSON file:

- Run the *banners* plugin on the image to determine the necessary kernel
 - Locate a copy of the debug kernel that matches the identified banner
 - Clone or update the dwarf2json repo: `git clone https://github.com/volatilityfoundation/dwarf2json`
 - Run `go build` in the directory if the source has changed
 - Run `dwarf2json linux --elf [path to debug kernel] > [kernel name].json`
 - For Mac change *linux* to *mac*
 - Copy the *.json* file to the symbols directory into *[symbols directory]/linux*
 - For Mac change *linux* to *mac*
-

CHANGES BETWEEN VOLATILITY 2 AND VOLATILITY 3

4.1 Library and Context

Volatility 3 has been designed from the ground up to be a library, this means the components are independent and all state required to run a particular plugin at a particular time is self-contained in an object derived from a *ContextInterface*.

The context contains the two core components that make up Volatility, layers of data and the available symbols.

4.2 Symbols and Types

Volatility 3 no longer uses profiles, it comes with an extensive library of *symbol tables*, and can generate new symbol tables for most windows memory images, based on the memory image itself. This allows symbol tables to include specific offsets for locations (symbol locations) based on that operating system in particular. This means it is easier and quicker to identify structures within an operating system, by having known offsets for those structures provided by the official debugging information.

4.3 Object Model changes

The object model has changed as well, objects now inherit directly from their Python counterparts, meaning an integer object is actually a Python integer (and has all the associated methods, and can be used wherever a normal int could). In Volatility 2, a complex proxy object was constructed which tried to emulate all the methods of the host object, but ultimately it was a different type and could not be used in the same places (critically, it could make the ordering of operations important, since $a + b$ might not work, but $b + a$ might work fine).

Volatility 3 has also had significant speed improvements, where Volatility 2 was designed to allow access to live memory images and situations in which the underlying data could change during the run of the plugin, in Volatility 3 the data is now read once at the time of object construction, and will remain static, even if the underlying layer changes. This was because live memory analysis was barely ever used, and this feature could cause a particular value to be re-read many times over for no benefit (particularly since each re-read could result in many additional image reads from following page table translations).

Finally, in order to provide Volatility specific information without impact on the ability for structures to have members with arbitrary names, all the metadata about the object (such as its layer or offset) have been moved to a read-only *vol()* dictionary.

Further the distinction between a *Template* (the thing that constructs an object) and the *Object* itself has been made more explicit. In Volatility 2, some information (such as size) could only be determined from a constructed object, leading to instantiating a template on an empty buffer, just to determine the size. In Volatility 3, templates contain information such as their size, which can be queried directly without constructing the object.

4.4 Layer and Layer dependencies

Address spaces in Volatility 2, are now more accurately referred to as *Translation Layers*, since each one typically sits atop another and can translate addresses between the higher logical layer and the lower physical layer. Address spaces in Volatility 2 were strictly limited to a stack, one on top of one other. In Volatility 3, layers can have multiple “dependencies” (lower layers), which allows for the integration of features such as swap space.

4.5 Automagic

In Volatility 2, we often tried to make this simpler for both users and developers. This resulted in something was referred to as automagic, in that it was magic that happened automatically. We’ve now codified that more, so that the automagic processes are clearly defined and can be enabled or disabled as necessary for any particular run. We also included a stacker automagic to emulate the most common feature of Volatility 2, automatically stacking address spaces (now translation layers) on top of each other.

By default the automagic chosen to be run are determined based on the plugin requested, so that linux plugins get linux specific automagic and windows plugins get windows specific automagic. This should reduce unnecessarily searching for linux kernels in a windows image, for example. At the moment this is not user configurable.

4.6 Searching and Scanning

Scanning is very similar to scanning in Volatility 2, a scanner object (such as a *BytesScanner* or *RegexScanner*) is primed with the data to be searched for, and the *scan()* method is called on the layer to be searched.

4.7 Output Rendering

This is extremely similar to Volatility 2, because we were developing it for Volatility 3 when we added it to Volatility 2. We now require that all plugins produce output in a *TreeGrid* object, which ensure that the library can be used regardless of which interface is driving it. An example web GUI is also available called Volumetric which allows all the plugins that can be run from the command line to be run from a webpage, and offers features such as automatic formatting and sorting of the data, which previously couldn’t be provided easily from the CLI.

There is also the ability to provide file output such that the user interface can provide a means to render or save those files.

VOLSHELL - A CLI TOOL FOR WORKING WITH MEMORY

Volshell is a utility to access the volatility framework interactively with a specific memory image. It allows for direct introspection and access to all features of the volatility library from within a command line environment.

5.1 Starting volshell

Volshell is started in much the same way as volatility. Rather than providing a plugin, you just specify the file. If the operating system of the memory image is known, a flag can be provided allowing additional methods for the specific operating system.

```
$ volshell.py -f <path-to-memory-image> [-w|-m|-l]
```

The flags to specify a known operating system are -w for windows, -m for mac and -l for linux. Volshell will run through the usual automagic, trying to load the memory image. If no operating system is specified, all automagic will be run.

When volshell starts, it will show the version of volshell, a brief message indicating how to get more help, the current operating system mode for volshell, and the current layer available for use.

```
Volshell (Volatility 3 Framework) 2.0.2
Readline imported successfully      PDB scanning finished

    Call help() to see available functions

Volshell mode      : Generic
Current Layer      : primary
Current Symbol Table : None
Current Kernel Name : None

(primary) >>>
```

Volshell itself is essentially a plugin, but an interactive one. As such, most values are accessed through *self* although there is also a *context* object whenever a context must be provided.

The prompt for the tool will indicate the name of the current layer (which can be accessed as *self.current_layer* from within the tool).

The generic mode is quite limited, won't have any symbols loaded and therefore won't be able to display much information. When an operating system is chosen, the appropriate symbols should be loaded and additional functions become available. The mode cannot easily be changed once the tool has started.

5.2 Accessing objects

All operating systems come with their equivalent of a process list, aliased to the function `ps()`. Running this will provide a list of volatility objects, based on the operating system in question. We will use these objects to run our examples against.

We'll start by creating a process variable, and putting the first result from `ps()` in it. Since the shell is a python environment, we can do the following:

```
(layer_name) >>> proc = ps()[0]
(layer_name) >>> proc
<EPROCESS symbol_table_name1!_EPROCESS: layer_name @ 0xe08ff2459040 #1968>
```

When printing a volatility structure, various information is output, in this case the *type_name*, the *layer* and *offset* that it's been constructed on, and the size of the structure.

We can directly access the volatility information about a structure, using the *.vol* attribute, which contains basic information such as structure size, *type_name*, and the list of members amongst others. However, volshell has a built-in mechanism for providing more information about a structure, called *display_type* or *dt*. This can be given either a type name (which if not prefixed with symbol table name, will use the kernel symbol table identified by the automagic).

```
(layer_name) >>> dt('_EPROCESS')
symbol_table_name1!_EPROCESS (1968 bytes)
    0x0 : Pcb                                symbol_table_name1!_KPROCESS
    0x2d8 : ProcessLock                      symbol_table_name1!_EX_PUSH_LOCK
    0x2e0 : RundownProtect                  symbol_table_name1!_EX_RUNDOWN_REF
    0x2e8 : UniqueProcessId                symbol_table_name1!pointer
...
```

It can also be provided with an object and will interpret the data for each in the process:

```
(layer_name) >>> dt(proc)
symbol_table_name1!_EPROCESS (1968 bytes)
    0x0 : Pcb                                symbol_table_name1!_KPROCESS
    ↪                                0xe08ff2459040
    0x2d8 : ProcessLock                      symbol_table_name1!_EX_PUSH_LOCK
    ↪                                0xe08ff2459318
    0x2e0 : RundownProtect                  symbol_table_name1!_EX_RUNDOWN_
    ↪REF                                0xe08ff2459320
    0x2e8 : UniqueProcessId                symbol_table_name1!pointer
    ↪                                4
...
```

These values can be accessed directly as attributes

```
(layer_name) >>> proc.UniqueProcessId
356
```

Pointer structures contain the value they point to, but attributes accessed are forwarded to the object they point to. This means that pointers do not need to be explicitly dereferenced to access underlying objects.

```
(layer_name) >>> proc.Pcb.DirectoryTableBase
4355817472
```


5.3 Running plugins

It's possible to run any plugin by importing it appropriately and passing it to the `display_plugin_output` or `dpo` method. In the following example we'll provide no additional parameters. Volatility will show us which parameters were required:

```
(layer_name) >>> from volatility3.plugins.windows import pslist
(layer_name) >>> display_plugin_output(pslist.PsList)
Unable to validate the plugin requirements: ['plugins.Volshell.
↳ VH3FSA1JBG0QP9E62Z80T5UCIMLNYKW4.PsList.kernel']
```

We can see that it's made a temporary configuration path for the plugin, and that the `kernel` requirement was not fulfilled.

We can see all the options that the plugin can accept by access the `get_requirements()` method of the plugin. This is a classmethod, so can be called on an uninstantiated copy of the plugin.

```
(layer_name) >>> pslist.PsList.get_requirements()
[<ModuleRequirement: kernel>, <BooleanRequirement: physical>, <ListRequirement: pid>,
↳ <BooleanRequirement: dump>]
```

We can provide arguments via the `dpo` method call:

```
(layer_name) >>> display_plugin_output(pslist.PsList, kernel = self.config['kernel'])
```

PID	PPID	ImageFileName	Offset(V)	Threads	Handles	SessionId	Wow64
↳ CreateTime		ExitTime	File output				
4	0	System	0x8c0bcac87040	143	-	N/A	False
↳ 0000000		N/A	Disabled				
92	4	Registry	0x8c0bcac5d080	4	-	N/A	False
↳ 17:25:28.000000		N/A	Disabled				
356	4	smss.exe	0x8c0bccf8d040	3	-	N/A	False
↳ 17:25:33.000000		N/A	Disabled				
...							

Here's we've provided the kernel name that was requested by the volshell plugin itself (the generic volshell does not load a kernel module, and instead only has a TranslationLayerRequirement). A different module could be created and provided instead. The context used by the `dpo` method is always `context`.

Instead of print the results directly to screen, they can be gathered into a TreeGrid objects for direct access by using the `generate_treegrid` or `gt` command.

```
(layer_name) >>> treegrid = gt(pslist.PsList, kernel = self.config['kernel'])
(layer_name) >>> treegrid.populate()
```

Treegrids must be populated before the data in them can be accessed. This is where the plugin actually runs and produces data.

5.4 Running scripts

It might be beneficial to code up a small snippet of code, and execute that on a memory image, rather than writing a full plugin.

The snippet should be lines that will be executed within the volshell context (as such they can immediately access *self* and *context*, for example). These can be executed using the *run_script* or *rs* command, or by providing the file on the command line with *-script*.

For example, to load a layer and extract bytes from a particular offset into a new file, the following snippet could be used:

```
import volatility3.framework.layers.mynewlayer as mynewlayer

layer = cc(mynewlayer.MyNewLayer, on_top_of = 'primary', other_parameter = 'important')
with open('output.dmp', 'wb') as fp:
    for i in range(0, 1073741824, 0x1000):
        data = layer.read(i, 0x1000, pad = True)
        fp.write(data)
```

As this demonstrates, all of the python is accessible, as are the volshell built in functions (such as *cc* which creates a constructable, like a layer or a symbol table).

5.5 Loading files

Files can be loaded as physical layers using the *load_file* or *lf* command, which takes a filename or a URI. This will be added to *context.layers* and can be accessed by the name returned by *lf*.

GLOSSARY

There are many terms when talking about memory forensics, this list hopes to define the common ones and provide some commonality on how to refer to particular ideas within the field.

6.1 A

Address Space

This is the name in volatility 2 for what's referred to as a *Translation Layer*. It encompasses all values that can be addresses, usually in reference to addresses in memory.

Alignment

This value is what all data *offsets* will typically be a multiple of within a *type*.

Array

This represents a list of items, which can be access by an index, which is zero-based (meaning the first element has index 0). Items in arrays are almost always the same size (it is not a generic list, as in python) even if they are *pointers* to different sized objects.

6.2 D

Data Layer

A group of bytes, where each byte can be addressed by a specific offset. Data layers are usually contiguous chunks of data.

Dereference

The act of taking the value of a pointer, and using it as an offset to another object, as a reference.

Domain

This the grouping for input values for a mapping or mathematical function.

6.3 M

Map, mapping

A mapping is a relationship between two sets (where elements of the *Domain* map to elements of the *Range*). Mappings can be seen as a mathematical function, and therefore volatility 3 attempts to use mathematical functional notation where possible. Within volatility a mapping is most often used to refer to the function for translating addresses from a higher layer (domain) to a lower layer (range). For further information, please see *Function (mathematics)* in wikipedia [https://en.wikipedia.org/wiki/Function_\(mathematics\)](https://en.wikipedia.org/wiki/Function_(mathematics))

Member

The name of subcomponents of a type, similar to attributes of objects in common programming parlance. These are usually recorded as *offset* and *type* pairs within a *structure*.

6.4 O

Object

This has a specific meaning within computer programming (as in Object Oriented Programming), but within the world of Volatility it is used to refer to a type that has been associated with a chunk of data, or a specific instance of a type. See also *Type*.

Offset

A numeric value that identifies a distance within a group of bytes, to uniquely identify a single byte, or the start of a run of bytes. An offset is often relative (offset from another object/item) but can be absolute (offset from the start of a region of data).

6.5 P

Packed

Structures are often *aligned* meaning that the various members (subtypes) are always aligned at particular values (usually multiples of 2, 4 or 8). Thus if the data used to represent a particular value has an odd number of bytes, not a multiple of the chosen number, there will be *padding* between it and the next member. In packed structs, no padding is used and the offset of the next member depends on the length of the previous one.

Padding

Data that (usually) contains no useful information. The typical value used for padding is 0 (sometimes called a null byte). As an example, if a string *object* that has been allocated a particular number of bytes, actually contains fewer bytes, the rest of the data (to make up the original length) will be padded with null (0) bytes.

Page

A specific chunk of contiguous data. It is an organizational quantity of memory (usually 0x1000, or 4096 bytes). Pages, like pages in a book, make up the whole, but allow for specific chunks to be allocated and used as necessary. Operating systems uses pages as a means to have granular control over chunks of memory. This allows them to be reordered and reused as necessary (without having to move large chunks of data around), and allows them to have access controls placed upon them, limiting actions such as reading and writing.

Page Table

A table that points to a series of *pages*. Each page table is typically the size of a single page, and page tables can point to pages that are in fact other page tables. Using tables that point to tables, it's possible to use them as a way to map a particular address within a (potentially larger, but sparsely populated) virtual space to a concrete (and usually contiguous) physical space, through the process of *mapping*.

Pointer

A value within memory that points to a different area of memory. This allows objects to contain references to

other objects without containing all the data of the other object. Following a pointer is known as *dereferencing* a pointer. Pointers are usually the same length as the maximum address of the address space, since they should be able to point to any address within the space.

6.6 R

Range

This is the set of the possible output values for a mapping or mathematical function.

6.7 S

Struct, Structure

A means of containing multiple different *type* associated together. A struct typically contains other *type*, usually *aligned* (unless *packing* is involved). In this way the *members* of a type can be accessed by finding the data at the relative *offset* to the start of the structure.

Symbol

This is used in many different contexts, as a short term for many things. Within Volatility, a symbol is a construct that usually encompasses a specific *type* at a specific *offset*, representing a particular instance of that type within the memory of a compiled and running program. An example would be the location in memory of a list of active tcp endpoints maintained by the networking stack within an operating system.

6.8 T

Template

Within volatility 3, the term template applies to a *type* that has not yet been instantiated or linked to any data or a specific location within memory. Once a type has been tied to a particular chunk of data, it is called an *object*.

Translation Layer

This is a type of data layer which allows accessing data from lower layers using addresses different to those used by the lower layers themselves. When accessing data in a translation layer, it translates (or *maps*) addresses from its own *address space* to the address space of the lower layer and returns the corresponding data from the lower layer. Note that multiple addresses in the higher layer might refer to the same address in the lower layer. Conversely, some addresses in the higher layer might have no corresponding address in the lower layer at all. Translation layers most commonly handle the translation from virtual to physical addresses, but can be used to translate data to and from a compressed form or translate data from a particular file format into another format.

Type

This is a structure definition of multiple elements that expresses how data is laid out. Basic types define how the data should be interpreted in terms of a run of bits (or more commonly a collection of 8 bits at a time, called bytes). New types can be constructed by combining other types at specific relative offsets, forming something called a *struct*, or by repeating the same type, known as an *array*. They can even contain other types at the same offset depending on the data itself, known as *Unions*. Once a type has been linked to a specific chunk of data, the result is referred to as an *object*.

6.9 U

Union

A union is a type that can hold multiple different subtypes, whose relative offsets specifically overlap. A union is a means for holding multiple different types within the same size of data, the relative offsets of the types within the union specifically overlap. This means that the data in a union object is interpreted differently based on the types of the union used to access it.

There is also some information to get you started quickly:

LINUX TUTORIAL

This guide will give you a brief overview of how volatility3 works as well as a demonstration of several of the plugins available in the suite.

7.1 Acquiring memory

Volatility3 does not provide the ability to acquire memory. Below are some examples of tools that can be used to acquire memory, but more are available:

- [AVML](#) - Acquire Volatile Memory for Linux
- [LiME](#) - Linux Memory Extract

7.2 Procedure to create symbol tables for linux

To create a symbol table please refer to *Mac or Linux symbol tables*.

Tip: It may be possible to locate pre-made ISF files from the [Linux ISF Server](#), which is built and maintained by [kevthehermit](#). After creating the file or downloading it from the ISF server, place the file under the directory `volatility3/symbols/linux`. If necessary create a linux directory under the symbols directory (this will become unnecessary in future versions).

7.3 Listing plugins

The following is a sample of the linux plugins available for volatility3, it is not complete and more more plugins may be added. For a complete reference, please see the volatility 3 *list of plugins*. For plugin requests, please create an issue with a description of the requested plugin.

```
$ python3 vol.py --help | grep -i linux. | head -n 5
banners.Banners      Attempts to identify potential linux banners in an
linux.bash.Bash       Recovers bash command history from memory.
linux.check_afinfo.Check_afinfo
linux.check_creds.Check_creds
linux.check_idt.Check_idt
```

Note: Here the the command is piped to grep and head in-order to provide the start of the list of linux plugins.

7.4 Using plugins

The following is the syntax to run the volatility CLI.

```
$ python3 vol.py -f <path to memory image> <plugin_name> <plugin_option>
```

7.5 Example

7.5.1 banners

In this example we will be using a memory dump from the Insomni'hack teaser 2020 CTF Challenge called Getdents. We will limit the discussion to memory forensics with volatility 3 and not extend it to other parts of the challenge. Thanks go to [stuxnet](#) for providing this memory dump and [writeup](#).

```
$ python3 vol.py -f memory.vmem banners

Volatility 3 Framework 2.0.1

Progress: 100.00          PDB scanning finished
Offset  Banner

0x141c1390      Linux version 4.15.0-42-generic (bulld@lgw01-amd64-023) (gcc_
↪version 7.3.0 (Ubuntu 7.3.0-16ubuntu3)) #45-Ubuntu SMP Thu Nov 15 19:32:57 UTC 2018_
↪(Ubuntu 4.15.0-42.45-generic 4.15.18)
0x63a00160      Linux version 4.15.0-72-generic (bulld@lcy01-amd64-026) (gcc_
↪version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)) #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC_
↪2019 (Ubuntu 4.15.0-72.81-generic 4.15.18)
0x6455c4d4      Linux version 4.15.0-72-generic (bulld@lcy01-amd64-026) (gcc_
↪version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)) #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC_
↪2019 (Ubuntu 4.15.0-72.81-generic 4.15.18)
0x6e1e055f      Linux version 4.15.0-72-generic (bulld@lcy01-amd64-026) (gcc_
↪version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)) #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC_
↪2019 (Ubuntu 4.15.0-72.81-generic 4.15.18)
0x7fde0010      Linux version 4.15.0-72-generic (bulld@lcy01-amd64-026) (gcc_
↪version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)) #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC_
↪2019 (Ubuntu 4.15.0-72.81-generic 4.15.18)
```

The above command helps us to find the memory dump's kernel version and the distribution version. Now using the above banner we can search for the needed ISF file from the ISF server. If ISF file cannot be found then, follow the instructions on [Procedure to create symbol tables for linux](#). After that, place the ISF file under the volatility3/symbols/linux directory.

Tip: Use the banner text which is most repeated to search from ISF Server.

7.5.2 linux.pslist

```
$ python3 vol.py -f memory.vmem linux.pslist

Volatility 3 Framework 2.0.1    Stacking attempts finished

PID      PPID     COMM
1         0        systemd
2         0        kthreadd
3         2        kworker/0:0
4         2        kworker/0:0H
5         2        kworker/u256:0
6         2        mm_percpu_wq
7         2        ksoftirqd/0
8         2        rcu_sched
9         2        rcu_bh
10        2        migration/0
11        2        watchdog/0
12        2        cpuhp/0
13        2        kdevtmpfs
14        2        netns
15        2        rcu_tasks_kthre
16        2        kauditd
.....
```

linux.pslist helps us to list the processes which are running, their PIDs and PPIDs.

7.5.3 linux.pstree

```
$ python3 vol.py -f memory.vmem linux.pstree

Volatility 3 Framework 2.0.1
Progress: 100.00    Stacking attempts finished

PID      PPID     COMM
1         0        systemd
* 636    1        polkitd
* 514    1        acpid
* 1411   1        pulseaudio
* 517    1        rsyslogd
* 637    1        cups-browsed
* 903    1        whoopsie
* 522    1        ModemManager
* 525    1        cron
* 526    1        avahi-daemon
** 542   526     avahi-daemon
* 657    1        unattended-upgr
* 914    1        kerneloops
* 532    1        dbus-daemon
* 1429   1        ibus-x11
* 929    1        kerneloops
* 1572   1        gsd-printer
```

(continues on next page)

(continued from previous page)

```

* 933 1 upowerd
* 1071 1 rtkit-daemon
* 692 1 gdm3
** 1234 692 gdm-session-wor
*** 1255 1234 gdm-x-session
**** 1257 1255 Xorg
**** 1266 1255 gnome-session-b
***** 1537 1266 gsd-clipboard
***** 1539 1266 gsd-color
***** 1542 1266 gsd-datetime
***** 2950 1266 deja-dup-monito
***** 1546 1266 gsd-housekeepin
***** 1548 1266 gsd-keyboard
***** 1550 1266 gsd-media-keys

```

`linux.pstree` helps us to display the parent child relationships between processes.

7.5.4 linux.bash

Now to find the commands that were run in the bash shell by using `linux.bash`.

```
$ python3 vol.py -f memory.vmem linux.bash
```

```

Volatility 3 Framework 2.0.1
Progress: 100.00 Stacking attempts finished
PID Process CommandTime Command

1733 bash 2020-01-16 14:00:36.000000 sudo reboot
1733 bash 2020-01-16 14:00:36.000000 AWAVH
1733 bash 2020-01-16 14:00:36.000000 sudo apt upgrade
1733 bash 2020-01-16 14:00:36.000000 sudo apt upgrade
1733 bash 2020-01-16 14:00:36.000000 sudo reboot
1733 bash 2020-01-16 14:00:36.000000 sudo apt update
1733 bash 2020-01-16 14:00:36.000000 sudo apt update
1733 bash 2020-01-16 14:00:36.000000 sudo reboot
1733 bash 2020-01-16 14:00:36.000000 sudo apt upgrade
1733 bash 2020-01-16 14:00:36.000000 sudo apt update
1733 bash 2020-01-16 14:00:36.000000 rub
1733 bash 2020-01-16 14:00:36.000000 sudo apt upgrade
1733 bash 2020-01-16 14:00:36.000000 uname -a
1733 bash 2020-01-16 14:00:36.000000 uname -a
1733 bash 2020-01-16 14:00:36.000000 sudo apt autoclean
1733 bash 2020-01-16 14:00:36.000000 sudo reboot
1733 bash 2020-01-16 14:00:36.000000 sudo apt upgrade
1733 bash 2020-01-16 14:00:41.000000 chmod +x meterpreter
1733 bash 2020-01-16 14:00:42.000000 sudo ./meterpreter

```

MACOS TUTORIAL

This guide will give you a brief overview of how volatility3 works as well as a demonstration of several of the plugins available in the suite.

8.1 Acquiring memory

Volatility3 does not provide the ability to acquire memory. The example below is an open source tool. Other commercial tools are also available.

- `osxpmem`

8.2 Procedure to create symbol tables for macOS

To create a symbol table please refer to *Mac or Linux symbol tables*.

Tip: It may be possible to locate pre-made ISF files from the [download link](#), which is built and maintained by [volatilityfoundation](#). After creating the file or downloading it from the link, place the file under the directory `volatility3/symbols/`.

8.3 Listing plugins

The following is a sample of the macOS plugins available for volatility3, it is not complete and more plugins may be added. For a complete reference, please see the volatility 3 *list of plugins*. For plugin requests, please create an issue with a description of the requested plugin.

```
$ python3 vol.py --help | grep -i mac. | head -n 4
mac.bash.Bash          Recovers bash command history from memory.
mac.check_syscall.Check_syscall
mac.check_sysctl.Check_sysctl
mac.check_trap_table.Check_trap_table
```

Note: Here the the command is piped to `grep` and `head` in-order to provide the start of the list of macOS plugins.

8.4 Using plugins

The following is the syntax to run the volatility CLI.

```
$ python3 vol.py -f <path to memory image> <plugin_name> <plugin_option>
```

8.5 Example

8.5.1 banners

In this example we will be using a memory dump from the Securinets CTF Quals 2019 Challenge called `Contact_me`. We will limit the discussion to memory forensics with volatility 3 and not extend it to other parts of the challenge. Thanks go to [stuxnet](#) for providing this memory dump and [writeup](#).

```
$ python3 vol.py -f contact_me banners.Banners

Volatility 3 Framework 2.4.2

Progress: 100.00          PDB scanning finished
Offset  Banner

0x4d2c7d0      Darwin Kernel Version 16.7.0: Thu Jun 15 17:36:27 PDT 2017; root:xnu-
↪3789.70.16~2/RELEASE_ARM64_T8020
0xb42b180      Darwin Kernel Version 16.7.0: Thu Jun 15 17:36:27 PDT 2017; root:xnu-
↪3789.70.16~2/RELEASE_ARM64_T8020
0xcda9100      Darwin Kernel Version 16.7.0: Thu Jun 15 17:36:27 PDT 2017; root:xnu-
↪3789.70.16~2/RELEASE_ARM64_T8020
0x1275e7d0     Darwin Kernel Version 16.7.0: Thu Jun 15 17:36:27 PDT 2017; root:xnu-
↪3789.70.16~2/RELEASE_ARM64_T8020
0x1284fba4     Darwin Kernel Version 16.7.0: Thu Jun 15 17:36:27 PDT 2017; root:xnu-
↪3789.70.16~2/RELEASE_ARM64_T8020
0x34ad0180     Darwin Kernel Version 16.7.0: Thu Jun 15 17:36:27 PDT 2017; root:xnu-
↪3789.70.16~2/RELEASE_ARM64_T8020
```

The above command helps us to find the memory dump's Darwin kernel version. Now using the above banner we can search for the needed ISF file. If ISF file cannot be found then, follow the instructions on [Procedure to create symbol tables for macOS](#). After that, place the ISF file under the `volatility3/symbols` directory.

8.5.2 mac.pslist

```
$ python3 vol.py -f contact_me mac.pslist.PsList

Volatility 3 Framework 2.4.2
Progress: 100.00          Stacking attempts finished

PID      PPID      COMM
0         0         kernel_task
1         0         launchd
```

(continues on next page)

(continued from previous page)

```

35      1      UserEventAgent
38      1      kextd
39      1      fsevents
37      1      uninstalld
45      1      configd
46      1      powerd
52      1      logd
58      1      warmd
.....

```

`mac.pslist` helps us to list the processes which are running, their PIDs and PPIDs.

8.5.3 mac.pstree

```

$ python3 vol.py -f contact_me mac.pstree.PsTree
Volatility 3 Framework 2.4.2
Progress: 100.00          Stacking attempts finished
PID      PPID      COMM
35        1        UserEventAgent
38        1        kextd
39        1        fsevents
37        1        uninstalld
204       1        softwareupdated
* 449     204       SoftwareUpdateCo
337       1        system_installd
* 455     337       update_dyld_shar

```

`mac.pstree` helps us to display the parent child relationships between processes.

8.5.4 mac.ifconfig

```

$ python3 vol.py -f contact_me mac.ifconfig.Ifconfig

Volatility 3 Framework 2.4.2
Progress: 100.00          Stacking attempts finished
Interface      IP Address      Mac Address      Promiscuous
lo0
lo0      127.0.0.1      False
lo0      ::1            False
lo0      fe80:1::1      False
gif0
stf0
en0      00:0C:29:89:8B:F0      00:0C:29:89:8B:F0      False
en0      fe80:4::10fb:c89d:217f:52ae      00:0C:29:89:8B:F0      False
en0      192.168.140.128      00:0C:29:89:8B:F0      False
utun0
utun0      fe80:5::2a95:bb15:87e3:977c      False

```

we can use the `mac.ifconfig` plugin to get information about the configuration of the network interfaces of the host under investigation.

WINDOWS TUTORIAL

This guide provides a brief introduction to how volatility3 works as a demonstration of several of the plugins available in the suite.

9.1 Acquiring memory

Volatility does not provide the ability to acquire memory. Memory can be acquired using a number of tools, below are some examples but others exist:

- WinPmem
- FTK Imager

9.2 Listing Plugins

The following is a sample of the windows plugins available for volatility3, it is not complete and more more plugins may be added. For a complete reference, please see the volatility 3 *list of plugins*. For plugin requests, please create an issue with a description of the requested plugin.

```
$ python3 vol.py --help | grep windows | head -n 5
windows.bigpools.BigPools
windows.cmdline.CmdLine
windows.crashinfo.Crashinfo
windows.dlllist.DllList
```

Note: Here the the command is piped to grep and head in-order to provide the start of a list of the available windows plugins.

9.3 Using plugins

The following is the syntax to run the volatility CLI.

```
$ python3 vol.py -f <path to memory image> plugin_name plugin_option
```

9.4 Example

9.4.1 windows.pslist

In this example we will be using a memory dump from the PragyanCTF'22. We will limit the discussion to memory forensics with volatility 3 and not extend it to other parts of the challenges.

When using windows plugins in volatility 3, the required ISF file can often be generated from PDB files automatically downloaded from Microsoft servers, and therefore does not require locating or adding specific ISF files to the volatility 3 symbols directory.

```
$ python3 vol.py -f MemDump.DMP windows.pslist | head -n 10
```

Volatility 3 Framework 2.0.1 PDB scanning finished									
PID	PPID	ImageFileName	Offset(V)	Threads	Handles	SessionId			
↳ Wow64	CreateTime	ExitTime		File output					
4	0	System	0xfa8000cbc040	85		492	N/A		
↳ False	2022-02-07	16:30:12.000000		N/A	Disabled				
276	4	smss.exe	0xfa8001e04040	2		29	N/A		
↳ False	2022-02-07	16:30:12.000000		N/A	Disabled				
352	336	csrss.exe	0xfa8002110b30	9		375	0		
↳ False	2022-02-07	16:30:13.000000		N/A	Disabled				
404	336	wininit.exe	0xfa800219f060	3		74	0		
↳ False	2022-02-07	16:30:13.000000		N/A	Disabled				
412	396	csrss.exe	0xfa80021c5b30	9		224	1		
↳ False	2022-02-07	16:30:13.000000		N/A	Disabled				
468	396	winlogon.exe	0xfa8002284060	5		113	1		
↳ False	2022-02-07	16:30:14.000000		N/A	Disabled				

windows.pslist helps list the processes running while the memory dump was taken.

9.4.2 windows.pstree

```
$ python3 vol.py -f MemDump.DMP windows.pstree | head -n 20
```

Volatility 3 Framework 2.0.1 PDB scanning finished									
PID	PPID	ImageFileName	Offset(V)	Threads	Handles	SessionId			
↳ Wow64	CreateTime	ExitTime							
4	0	System	0xfa8000cbc040	85	492	N/A	False	2022-02-07	↳
↳ 16:30:12.000000		N/A							
* 276	4	smss.exe	0xfa8001e04040	2	29	N/A	False	2022-	

(continues on next page)

(continued from previous page)

→02-07 16:30:12.000000	N/A								
352	336	csrss.exe	0xfa8002110b30	9	375	0	False	2022-	
→02-07 16:30:13.000000	N/A								
404	336	wininit.exe	0xfa800219f060	3	74	0	False	2022-	
→02-07 16:30:13.000000	N/A								
* 504	404	services.exe	0xfa80022ccb30	7	190	0	False	2022-	
→02-07 16:30:14.000000	N/A								
** 960	504	svchost.exe	0xfa8001c17b30	39	1003	0	False	2022-	
→02-07 16:30:14.000000	N/A								
** 1216	504	svchost.exe	0xfa80026e0b30	18	311	0	False	2022-	
→02-07 16:30:15.000000	N/A								
** 1312	504	svchost.exe	0xfa8002740380	19	287	0	False	2022-	
→02-07 16:30:15.000000	N/A								
** 1984	504	taskhost.exe	0xfa8002eb1b30	8	129	1	False	2022-	
→02-07 16:30:27.000000	N/A								
** 804	504	svchost.exe	0xfa80024ca5f0	20	450	0	False	2022-	
→02-07 16:30:14.000000	N/A								
*** 100	804	audiodg.exe	0xfa80025b4b30	6	131	0	False	2022-	
→02-07 16:30:14.000000	N/A								
** 1568	504	SearchIndexer.	0xfa800254b480	12	616	0	False	2022-	
→02-07 16:30:32.000000	N/A								
** 744	504	svchost.exe	0xfa8002477b30	8	265	0	False	2022-	
→02-07 16:30:14.000000	N/A								
** 1096	504	svchost.exe	0xfa800260db30	14	357	0	False	2022-	
→02-07 16:30:14.000000	N/A								
** 616	504	svchost.exe	0xfa8002b86ab0	13	314	0	False	2022-	
→02-07 16:32:16.000000	N/A								
** 624	504	svchost.exe	0xfa8002410630	10	350	0	False	2022-	
→02-07 16:30:14.000000	N/A								

windows.pstree helps to display the parent child relationships between processes.

Note: Here the the command is piped to head in-order to provide smaller output, here listing only the first 20.

9.4.3 windows.hashdump

```
$ python3 vol.py -f MemDump.DMP windows.hashdump
Volatility 3 Framework 2.0.3
Progress: 100.00          PDB scanning finished
User      rid      lmhash  nthash

Administrator      500      aad3b435b51404eeaad3b435b51404ee  ↪
→31d6cfe0d16ae931b73c59d7e0c089c0
Guest                501      aad3b435b51404eeaad3b435b51404ee  ↪
→31d6cfe0d16ae931b73c59d7e0c089c0
Frank Reynolds      1000     aad3b435b51404eeaad3b435b51404ee  ↪
→a88d1e18706d3aa676e01e5943d15911
HomeGroupUser$      1002     aad3b435b51404eeaad3b435b51404ee  ↪
→af10ecac6ea817d2bb56e3e5c33ce1cd
```

(continues on next page)

(continued from previous page)

```
Dennis          1003      aad3b435b51404eeaad3b435b51404ee  
↪cf96684bbc7877920adaa9663698bf54
```

`windows.hashdump` helps to list the hashes of the users in the system.

VOLATILITY3 PACKAGE

Volatility 3 - An open-source memory forensics framework

class `WarningFindSpec`

Bases: `MetaPathFinder`

Checks import attempts and throws a warning if the name shouldn't be used.

find_module(*fullname*, *path*)

Return a loader for the module.

If no module is found, return None. The fullname is a str and the path is a list of strings or None.

This method is deprecated since Python 3.4 in favor of `finder.find_spec()`. If `find_spec()` exists then backwards-compatible functionality is provided for this method.

static find_spec(*fullname*, *path*, *target=None*, ***kwargs*)

Mock `find_spec` method that just checks the name, this must go first.

Return type

`None`

invalidate_caches()

An optional method for clearing the finder's cache, if any. This method is used by `importlib.invalidate_caches()`.

class `classproperty`(*func*)

Bases: `property`

Class property decorator.

Note this will change the return type

deleter()

Descriptor to obtain a copy of the property with a different deleter.

fdel

fget

fset

getter()

Descriptor to obtain a copy of the property with a different getter.

setter()

Descriptor to obtain a copy of the property with a different setter.

10.1 Subpackages

10.1.1 volatility3.cli package

A CommandLine User Interface for the volatility framework.

User interfaces make use of the framework to:

- determine available plugins
- request necessary information for those plugins from the user
- determine what “automagic” modules will be used to populate information the user does not provide
- run the plugin
- display the results

class `CommandLine`

Bases: `object`

Constructs a command-line interface object for users to run plugins.

CLI_NAME = 'volatility'

file_handler_class_factory(*direct=True*)

classmethod `location_from_file(filename)`

Returns the URL location from a file parameter (which may be a URL)

Parameters

filename (`str`) – The path to the file (either an absolute, relative, or URL path)

Return type

`str`

Returns

The URL for the location of the file

populate_config(*context, configurables_list, args, plugin_config_path*)

Populate the context config based on the returned args.

We have already determined these elements must be descended from ConfigurableInterface

Parameters

- **context** (`ContextInterface`) – The volatility3 context to operate on
- **configurables_list** (`Dict[str, Type[ConfigurableInterface]]`) – A dictionary of configurable items that can be configured on the plugin
- **args** (`Namespace`) – An object containing the arguments necessary
- **plugin_config_path** (`str`) – The path within the context’s config containing the plugin’s configuration

Return type

`None`

populate_requirements_argparse(*parser, configurable*)

Adds the plugin’s simple requirements to the provided parser.

Parameters

- **parser** (`Union[ArgumentParser, _ArgumentGroup]`) – The parser to add the plugin’s (simple) requirements to
- **configurable** (`Type[ConfigurableInterface]`) – The plugin object to pull the requirements from

process_exceptions(*excp*)

Provide useful feedback if an exception occurs during a run of a plugin.

process_unsatisfied_exceptions(*excp*)

Provide useful feedback if an exception occurs during requirement fulfillment.

run()

Executes the command line module, taking the system arguments, determining the plugin to run and then running it.

classmethod setup_logging()

class MuteProgress

Bases: `PrintedProgress`

A dummy progress handler that produces no output when called.

class PrintedProgress

Bases: `object`

A progress handler that prints the progress value and the description onto the command line.

main()

A convenience function for constructing and running the `CommandLine`’s run method.

Subpackages

volatility3.cli.volshell package

class VolShell

Bases: `CommandLine`

Program to allow interactive interaction with a memory image.

This allows a memory image to be examined through an interactive python terminal with all the volatility support calls available.

CLI_NAME = 'volshell'

file_handler_class_factory(*direct=True*)

classmethod location_from_file(*filename*)

Returns the URL location from a file parameter (which may be a URL)

Parameters

filename (`str`) – The path to the file (either an absolute, relative, or URL path)

Return type

`str`

Returns

The URL for the location of the file

populate_config(*context, configurables_list, args, plugin_config_path*)

Populate the context config based on the returned args.

We have already determined these elements must be descended from ConfigurableInterface

Parameters

- **context** (*ContextInterface*) – The volatility3 context to operate on
- **configurables_list** (*Dict[str, Type[ConfigurableInterface]]*) – A dictionary of configurable items that can be configured on the plugin
- **args** (*Namespace*) – An object containing the arguments necessary
- **plugin_config_path** (*str*) – The path within the context’s config containing the plugin’s configuration

Return type

None

populate_requirements_argparse(*parser, configurable*)

Adds the plugin’s simple requirements to the provided parser.

Parameters

- **parser** (*Union[ArgumentParser, _ArgumentGroup]*) – The parser to add the plugin’s (simple) requirements to
- **configurable** (*Type[ConfigurableInterface]*) – The plugin object to pull the requirements from

process_exceptions(*excp*)

Provide useful feedback if an exception occurs during a run of a plugin.

process_unsatisfied_exceptions(*excp*)

Provide useful feedback if an exception occurs during requirement fulfillment.

run()

Executes the command line module, taking the system arguments, determining the plugin to run and then running it.

classmethod setup_logging()

main()

A convenience function for constructing and running the CommandLine’s run method.

Submodules

volatility3.cli.volshell.generic module

class NullFileHandler(*preferred_name*)

Bases: BytesIO, *FileHandlerInterface*

Null FileHandler that swallows files whole without consuming memory

close()

Disable all I/O operations.

closed

True if the file is closed.

detach()

Disconnect this buffer from its underlying raw stream and return it.

After the raw stream has been detached, the buffer is in an unusable state.

fileno()

Returns underlying file descriptor if one exists.

OSError is raised if the IO object does not use a file descriptor.

flush()

Does nothing.

getbuffer()

Get a read-write view over the contents of the BytesIO object.

getvalue()

Retrieve the entire contents of the BytesIO object.

isatty()

Always returns False.

BytesIO objects are not connected to a TTY-like device.

property preferred_filename

The preferred filename to save the data to. Until this file has been written, this value may not be the final filename the data is written to.

read(*size=-1, /*)

Read at most size bytes, returned as a bytes object.

If the size argument is negative, read until EOF is reached. Return an empty bytes object at EOF.

read1(*size=-1, /*)

Read at most size bytes, returned as a bytes object.

If the size argument is negative or omitted, read until EOF is reached. Return an empty bytes object at EOF.

readable()

Returns True if the IO object can be read.

readall()

Read until EOF, using multiple read() call.

readinto(*buffer, /*)

Read bytes into buffer.

Returns number of bytes read (0 for EOF), or None if the object is set not to block and has no data to read.

readinto1(*buffer, /*)**readline(*size=-1, /*)**

Next line from the file, as a bytes object.

Retain newline. A non-negative size argument limits the maximum number of bytes to return (an incomplete line may be returned then). Return an empty bytes object at EOF.

readlines(*size=None, /*)

List of bytes objects, each a line from the file.

Call `readline()` repeatedly and return a list of the lines so read. The optional `size` argument, if given, is an approximate bound on the total number of bytes in the lines returned.

static sanitize_filename(*filename*)

Sanitizes the filename to ensure only a specific whitelist of characters is allowed through

Return type

`str`

seek(*pos, whence=0, /*)

Change stream position.

Seek to byte offset `pos` relative to position indicated by `whence`:

0 Start of stream (the default). `pos` should be ≥ 0 ; 1 Current position - `pos` may be negative; 2 End of stream - `pos` usually negative.

Returns the new absolute position.

seekable()

Returns True if the IO object can be seeked.

tell()

Current file position, an integer.

truncate(*size=None, /*)

Truncate the file to at most `size` bytes.

Size defaults to the current file position, as returned by `tell()`. The current file position is unchanged. Returns the new size.

writable()

Returns True if the IO object can be written.

write(*b*)

Dummy method

writelines(*lines*)

Dummy method

class Volshell(**args, **kwargs*)

Bases: [*PluginInterface*](#)

Shell environment to directly interact with a memory image.

Parameters

- **context** – The context that the plugin will operate within
- **config_path** – The path to configuration data within the context configuration data
- **progress_callback** – A callable that can provide feedback at progress points

build_configuration()

Constructs a `HierarchicalDictionary` of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***change_kernel**(*kernel_name=None*)**change_layer**(*layer_name=None*)

Changes the current default layer

change_symbol_table(*symbol_table_name=None*)

Changes the current_symbol_table

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

construct_locals()

Returns a dictionary listing the functions to be added to the environment.

Return type*List[Tuple[List[str], Any]]***property context:** *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

create_configurable(*clazz, **kwargs*)

Creates a configurable object, converting arguments to configuration

property current_kernel_name**property current_layer****property current_symbol_table****disassemble**(*offset, count=128, layer_name=None, architecture=None*)

Disassembles a number of instructions from the code at offset

display_bytes(*offset, count=128, layer_name=None*)

Displays byte values and ASCII characters

display_doublewords(*offset, count=128, layer_name=None*)

Displays double-word values (4 bytes) and corresponding ASCII characters

display_plugin_output(*plugin, **kwargs*)

Displays the output for a particular plugin (with keyword arguments)

Return type*None***display_quadwords**(*offset, count=128, layer_name=None*)

Displays quad-word values (8 bytes) and corresponding ASCII characters

display_symbols(*symbol_table=None*)

Prints an alphabetical list of symbols for a symbol table

display_type(*object, offset=None*)

Display Type describes the members of a particular object in alphabetical order

display_words(*offset*, *count=128*, *layer_name=None*)

Displays word values (2 bytes) and corresponding ASCII characters

generate_treegrid(*plugin*, ***kwargs*)

Generates a TreeGrid based on a specific plugin passing in kwarg configuration values

Return type

`Optional[TreeGrid]`

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

help(**args*)

Describes the available commands

property kernel

Returns the current kernel object

load_file(*location*)

Loads a file into a Filelayer and returns the name of the layer

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

random_string(*length=32*)

Return type

`str`

render_treegrid(*treegrid*, *renderer=None*)

Renders a treegrid as produced by generate_treegrid

Return type

`None`

run(*additional_locals=None*)

Runs the interactive volshell plugin.

Return type

`TreeGrid`

Returns

Return a TreeGrid but this is always empty since the point of this plugin is to run interactively

run_script(*location*)

Runs a python script within the context of volshell

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.cli.volshell.linux module

class Volshell(*args, **kwargs)

Bases: *Volshell*

Shell environment to directly interact with a linux memory image.

Parameters

- **context** – The context that the plugin will operate within
- **config_path** – The path to configuration data within the context configuration data
- **progress_callback** – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

change_kernel(*kernel_name=None*)

change_layer(*layer_name=None*)

Changes the current default layer

change_symbol_table(*symbol_table_name=None*)

Changes the current_symbol_table

change_task(*pid=None*)

Change the current process and layer, based on a process ID

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

construct_locals()

Returns a dictionary listing the functions to be added to the environment.

Return type

List[Tuple[List[str], Any]]

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

create_configurable(*clazz, **kwargs*)

Creates a configurable object, converting arguments to configuration

property current_kernel_name

property current_layer

property current_symbol_table

disassemble(*offset, count=128, layer_name=None, architecture=None*)

Disassembles a number of instructions from the code at offset

display_bytes(*offset, count=128, layer_name=None*)

Displays byte values and ASCII characters

display_doublewords(*offset, count=128, layer_name=None*)

Displays double-word values (4 bytes) and corresponding ASCII characters

display_plugin_output(*plugin, **kwargs*)

Displays the output for a particular plugin (with keyword arguments)

Return type

None

display_quadwords(*offset, count=128, layer_name=None*)

Displays quad-word values (8 bytes) and corresponding ASCII characters

display_symbols(*symbol_table=None*)

Prints an alphabetical list of symbols for a symbol table

display_type(*object, offset=None*)

Display Type describes the members of a particular object in alphabetical order

display_words(*offset, count=128, layer_name=None*)

Displays word values (2 bytes) and corresponding ASCII characters

generate_treegrid(*plugin, **kwargs*)

Generates a TreeGrid based on a specific plugin passing in kwarg configuration values

Return type

Optional[TreeGrid]

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

help(**args*)

Describes the available commands

property `kernel`

Returns the current kernel object

list_tasks()

Returns a list of task objects from the primary layer

load_file(*location*)

Loads a file into a Filelayer and returns the name of the layer

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property `open`

Returns a context manager and thus can be called like open

random_string(*length=32*)

Return type

str

render_treegrid(*treegrid, renderer=None*)

Renders a treegrid as produced by generate_treegrid

Return type

None

run(*additional_locals=None*)

Runs the interactive volshell plugin.

Return type

TreeGrid

Returns

Return a TreeGrid but this is always empty since the point of this plugin is to run interactively

run_script(*location*)

Runs a python script within the context of volshell

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod **unsatisfied**(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.cli.volshell.mac module

class **Volshell**(*args, **kwargs)

Bases: `Volshell`

Shell environment to directly interact with a mac memory image.

Parameters

- **context** – The context that the plugin will operate within
- **config_path** – The path to configuration data within the context configuration data
- **progress_callback** – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

change_kernel(*kernel_name=None*)

change_layer(*layer_name=None*)

Changes the current default layer

change_symbol_table(*symbol_table_name=None*)

Changes the current_symbol_table

change_task(*pid=None*)

Change the current process and layer, based on a process ID

property **config**: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

construct_locals()

Returns a dictionary listing the functions to be added to the environment.

Return type

`List[Tuple[List[str], Any]]`

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

create_configurable(*clazz*, *kwargs*)**

Creates a configurable object, converting arguments to configuration

property current_kernel_name

property current_layer

property current_symbol_table

disassemble(*offset*, *count*=128, *layer_name*=None, *architecture*=None)

Disassembles a number of instructions from the code at offset

display_bytes(*offset*, *count*=128, *layer_name*=None)

Displays byte values and ASCII characters

display_doublewords(*offset*, *count*=128, *layer_name*=None)

Displays double-word values (4 bytes) and corresponding ASCII characters

display_plugin_output(*plugin*, *kwargs*)**

Displays the output for a particular plugin (with keyword arguments)

Return type

`None`

display_quadwords(*offset*, *count*=128, *layer_name*=None)

Displays quad-word values (8 bytes) and corresponding ASCII characters

display_symbols(*symbol_table*=None)

Prints an alphabetical list of symbols for a symbol table

display_type(*object*, *offset*=None)

Display Type describes the members of a particular object in alphabetical order

display_words(*offset*, *count*=128, *layer_name*=None)

Displays word values (2 bytes) and corresponding ASCII characters

generate_treegrid(*plugin*, *kwargs*)**

Generates a TreeGrid based on a specific plugin passing in kwarg configuration values

Return type

`Optional[TreeGrid]`

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

help(args*)**

Describes the available commands

property kernel

Returns the current kernel object

list_tasks(*method=None*)

Returns a list of task objects from the primary layer

load_file(*location*)

Loads a file into a Filelayer and returns the name of the layer

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

random_string(*length=32*)**Return type**

str

render_treegrid(*treegrid, renderer=None*)

Renders a treegrid as produced by generate_treegrid

Return type

None

run(*additional_locals=None*)

Runs the interactive volshell plugin.

Return type

TreeGrid

Returns

Return a TreeGrid but this is always empty since the point of this plugin is to run interactively

run_script(*location*)

Runs a python script within the context of volshell

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.cli.volshell.windows module

class `Volshell(*args, **kwargs)`

Bases: `Volshell`

Shell environment to directly interact with a windows memory image.

Parameters

- **context** – The context that the plugin will operate within
- **config_path** – The path to configuration data within the context configuration data
- **progress_callback** – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

change_kernel(kernel_name=None)

change_layer(layer_name=None)

Changes the current default layer

change_process(pid=None)

Change the current process and layer, based on a process ID

change_symbol_table(symbol_table_name=None)

Changes the current_symbol_table

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

construct_locals()

Returns a dictionary listing the functions to be added to the environment.

Return type

`List[Tuple[List[str], Any]]`

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

create_configurable(*clazz*, *kwargs*)**

Creates a configurable object, converting arguments to configuration

property current_kernel_name

property current_layer

property current_symbol_table

disassemble(*offset*, *count*=128, *layer_name*=None, *architecture*=None)

Disassembles a number of instructions from the code at offset

display_bytes(*offset*, *count*=128, *layer_name*=None)

Displays byte values and ASCII characters

display_doublewords(*offset*, *count*=128, *layer_name*=None)

Displays double-word values (4 bytes) and corresponding ASCII characters

display_plugin_output(*plugin*, *kwargs*)**

Displays the output for a particular plugin (with keyword arguments)

Return type

`None`

display_quadwords(*offset*, *count*=128, *layer_name*=None)

Displays quad-word values (8 bytes) and corresponding ASCII characters

display_symbols(*symbol_table*=None)

Prints an alphabetical list of symbols for a symbol table

display_type(*object*, *offset*=None)

Display Type describes the members of a particular object in alphabetical order

display_words(*offset*, *count*=128, *layer_name*=None)

Displays word values (2 bytes) and corresponding ASCII characters

generate_treegrid(*plugin*, *kwargs*)**

Generates a TreeGrid based on a specific plugin passing in kwarg configuration values

Return type

`Optional[TreeGrid]`

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

help(args*)**

Describes the available commands

property kernel

Returns the current kernel object

list_processes()

Returns a list of EPROCESS objects from the primary layer

load_file(location)

Loads a file into a Filelayer and returns the name of the layer

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

random_string(length=32)**Return type**

str

render_treegrid(treegrid, renderer=None)

Renders a treegrid as produced by generate_treegrid

Return type

None

run(additional_locals=None)

Runs the interactive volshell plugin.

Return type

TreeGrid

Returns

Return a TreeGrid but this is always empty since the point of this plugin is to run interactively

run_script(location)

Runs a python script within the context of volshell

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

`version = (0, 0, 0)`

Submodules

volatility3.cli.text_renderer module

class `CLIRenderer`(*options=None*)

Bases: `Renderer`

Class to add specific requirements for CLI renderers.

Accepts an options object to configure the renderers.

abstract `get_render_options()`

Returns a list of rendering options.

Return type

`List[Any]`

`name = 'unnamed'`

abstract `render(grid)`

Takes a grid object and renders it based on the object's preferences.

Return type

`None`

`structured_output = False`

class `CSVRenderer`(*options=None*)

Bases: `CLIRenderer`

Accepts an options object to configure the renderers.

get_render_options()

Returns a list of rendering options.

`name = 'csv'`

render(grid)

Renders each row immediately to stdout.

Parameters

grid (`TreeGrid`) – The TreeGrid object to render

Return type

`None`

`structured_output = True`

```
class JsonLinesRenderer(options=None)
    Bases: JsonRenderer

    Accepts an options object to configure the renderers.

    get_render_options()
        Returns a list of rendering options.

        Return type
        List[Any]

    name = 'JSONL'

    output_result(outfd, result)
        Outputs the JSON results as JSON lines

    render(grid)
        Takes a grid object and renders it based on the object's preferences.

    structured_output = True
```

```
class JsonRenderer(options=None)
    Bases: CLIRenderer

    Accepts an options object to configure the renderers.

    get_render_options()
        Returns a list of rendering options.

        Return type
        List[Any]

    name = 'JSON'

    output_result(outfd, result)
        Outputs the JSON data to a file in a particular format

    render(grid)
        Takes a grid object and renders it based on the object's preferences.

    structured_output = True
```

```
class NoneRenderer(options=None)
    Bases: CLIRenderer

    Outputs no results

    Accepts an options object to configure the renderers.

    get_render_options()
        Returns a list of rendering options.

    name = 'none'

    render(grid)
        Takes a grid object and renders it based on the object's preferences.

        Return type
        None

    structured_output = False
```

class PrettyTextRenderer(*options=None*)

Bases: [CLIRenderer](#)

Accepts an options object to configure the renderers.

get_render_options()

Returns a list of rendering options.

name = 'pretty'

render(*grid*)

Renders each column immediately to stdout.

This does not format each line's width appropriately, it merely tab separates each field

Parameters

grid ([TreeGrid](#)) – The TreeGrid object to render

Return type

[None](#)

structured_output = False

tab_stop(*line*)

Return type

[str](#)

class QuickTextRenderer(*options=None*)

Bases: [CLIRenderer](#)

Accepts an options object to configure the renderers.

get_render_options()

Returns a list of rendering options.

name = 'quick'

render(*grid*)

Renders each column immediately to stdout.

This does not format each line's width appropriately, it merely tab separates each field

Parameters

grid ([TreeGrid](#)) – The TreeGrid object to render

Return type

[None](#)

structured_output = False

display_disassembly(*disasm*)

Renders a disassembly renderer type into string format.

Parameters

disasm ([Disassembly](#)) – Input disassembly objects

Return type

[str](#)

Returns

A string as rendered by capstone where available, otherwise output as if it were just bytes

hex_bytes_as_text(*value*)

Renders HexBytes as text.

Parameters

value (*bytes*) – A series of bytes to convert to text

Return type

str

Returns

A text representation of the hexadecimal bytes plus their ascii equivalents, separated by newline characters

multitypedata_as_text(*value*)

Renders the bytes as a string where possible, otherwise it displays hex data

This attempts to convert the string based on its encoding and if no data's been lost due to the split on the null character, then it displays it as is

Return type

str

optional(*func*)

Return type

Callable

quoted_optional(*func*)

Return type

Callable

volatility3.cli.volargparse module

```
class HelpfulArgParser(prog=None, usage=None, description=None, epilog=None, parents=[],
                      formatter_class=<class 'argparse.HelpFormatter'>, prefix_chars='-',
                      fromfile_prefix_chars=None, argument_default=None, conflict_handler='error',
                      add_help=True, allow_abbrev=True, exit_on_error=True)
```

Bases: *ArgumentParser*

add_argument(*dest, ..., name=value, ...*)

add_argument(*option_string, option_string, ..., name=value, ...*) → *None*

add_argument_group(**args, **kwargs*)

add_mutually_exclusive_group(***kwargs*)

add_subparsers(***kwargs*)

convert_arg_line_to_args(*arg_line*)

error(*message: string*)

Prints a usage message incorporating the message to stderr and exits.

If you override this in a subclass, it should not return – it should either exit or raise an exception.

exit(*status=0, message=None*)

```
format_help()
format_usage()
get_default(dest)
parse_args(args=None, namespace=None)
parse_intermixed_args(args=None, namespace=None)
parse_known_args(args=None, namespace=None)
parse_known_intermixed_args(args=None, namespace=None)
print_help(file=None)
print_usage(file=None)
register(registry_name, value, object)
set_defaults(**kwargs)
class HelpfulSubparserAction(*args, **kwargs)
    Bases: _SubParsersAction
    Class to either select a unique plugin based on a substring, or identify the alternatives.
    add_parser(name, **kwargs)
    format_usage()
```

10.1.2 volatility3.framework package

Volatility 3 framework.

```
class NonInheritable(value, cls)
    Bases: object
class_subclasses(cls)
    Returns all the (recursive) subclasses of a given class.
    Return type
        Generator[Type[TypeVar(T)], None, None]
clear_cache(complete=False)
hide_from_subclasses(cls)
    Return type
        Type
import_file(module, path, ignore_errors=False)
    Imports a python file based on an existing module, a submodule and a filepath for error messages
    Return type
        List[str]
Args
    module: Module name to be imported path: File to be imported from (used for error messages)
```


Returns

List of modules that may have failed to import

import_files(*base_module*, *ignore_errors=False*)

Imports all plugins present under plugins module namespace.

Return type

`List[str]`

interface_version()

Provides the so version number of the library.

Return type

`Tuple[int, int, int]`

list_plugins()

Return type

`Dict[str, Type[PluginInterface]]`

require_interface_version(**args*)

Checks the required version of a plugin.

Return type

`None`

Subpackages**volatility3.framework.automagic package**

Automagic modules allow the framework to populate configuration elements that a user has not provided.

Automagic objects accept a *context* and a *configurable*, and will make appropriate changes to the *context* in an attempt to fulfill the requirements of the *configurable* object (or objects upon which that configurable may rely).

Several pre-existing modules include one to stack layers on top of each other (allowing automatic detection and loading of file format types) as well as a module to reconstruct layers based on their provided requirements.

available(*context*)

Returns an ordered list of all subclasses of *AutomagicInterface*.

The order is based on the priority attributes of the subclasses, in order to ensure the automagics are listed in an appropriate order.

Parameters

context (*ContextInterface*) – The context that will contain any automagic configuration values.

Return type

`List[AutomagicInterface]`

choose_automagic(*automagics*, *plugin*)

Chooses which automagics to run, maintaining the order they were handed in.

Return type

`List[Type[AutomagicInterface]]`

run(*automagics*, *context*, *configurable*, *config_path*, *progress_callback=None*)

Runs through the list of *automagics* in order, allowing them to make changes to the context.

Parameters

- **automagics** (`List[AutomagicInterface]`) – A list of *AutomagicInterface* objects
- **context** (*ContextInterface*) – The context (that inherits from *ContextInterface*) for modification
- **configurable** (`Union[ConfigurableInterface, Type[ConfigurableInterface]]`) – An object that inherits from *ConfigurableInterface*
- **config_path** (`str`) – The path within the *context.config* for options required by the *configurable*
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A function that takes a percentage (and an optional description) that will be called periodically

Return type

`List[TracebackException]`

This is where any automagic is allowed to run, and alter the context in order to satisfy/improve all requirements

Returns a list of traceback objects that occurred during the autorun procedure

Note: The order of the *automagics* list is important. An *automagic* that populates configurations may be necessary for an *automagic* that populates the context based on the configuration information.

Submodules

volatility3.framework.automagic.construct_layers module

An automagic module to use configuration data to configure and then construct classes that fulfill the descendants of a *ConfigurableInterface*.

class ConstructionMagic(*context*, *config_path*, **args*, ***kwargs*)

Bases: *AutomagicInterface*

Constructs underlying layers.

Class to run through the requirement tree of the *ConfigurableInterface* and from the bottom of the tree upwards, attempt to construct all *ConstructableRequirementInterface* based classes.

Warning

This *automagic* should run first to allow existing configurations to have been constructed for use by later automagic

Basic initializer that allows configurables to access their own config settings.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

exclusion_list = []

A list of plugin categories (typically operating systems) which the plugin will not operate on

find_requirements(*context, config_path, requirement_root, requirement_type, shortcut=True*)

Determines if there is actually an unfulfilled *Requirement* waiting.

This ensures we do not carry out an expensive search when there is no need for a particular *Requirement*

Parameters

- **context** (*ContextInterface*) – Context on which to operate
- **config_path** (*str*) – Configuration path of the top-level requirement
- **requirement_root** (*RequirementInterface*) – Top-level requirement whose sub-requirements will all be searched
- **requirement_type** (*Union[Tuple[Type[RequirementInterface], ...], Type[RequirementInterface]*) – Type of requirement to find
- **shortcut** (*bool*) – Only returns requirements that live under unsatisfied requirements

Return type

List[Tuple[str, RequirementInterface]]

Returns

A list of tuples containing the *config_path*, *sub_config_path* and requirement identifying the unsatisfied *Requirements*

classmethod get_requirements()

Returns a list of *RequirementInterface* objects required by this object.

Return type

List[RequirementInterface]

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from *kwargs*.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

priority = 0

An ordering to indicate how soon this automagic should be run

classmethod **unsatisfied**(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

volatility3.framework.automagic.linux module

class **LinuxIntelStacker**

Bases: `StackerLayerInterface`

exclusion_list: `List[str] = ['mac', 'windows']`

The list operating systems/first-level plugin hierarchy that should exclude this stacker

classmethod **find_aslr**(*context*, *symbol_table*, *layer_name*, *progress_callback=None*)

Determines the offset of the actual DTB in physical space and its symbol offset.

Return type

`Tuple[int, int]`

classmethod **stack**(*context*, *layer_name*, *progress_callback=None*)

Attempts to identify linux within this layer.

Return type

`Optional[DataLayerInterface]`

stack_order = 35

The order in which to attempt stacking, the lower the earlier

classmethod **stacker_slow_warning**()

classmethod **virtual_to_physical_address**(*addr*)

Converts a virtual linux address to a physical one (does not account of ASLR)

Return type

`int`

class **LinuxSymbolFinder**(*context*, *config_path*)

Bases: `SymbolFinder`

Linux symbol loader based on uname signature strings.

Basic initializer that allows configurables to access their own config settings.

banner_config_key: `str = 'kernel_banner'`

property **banners:** `Dict[bytes, List[str]]`

Creates a cached copy of the results, but only it's been requested.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

exclusion_list = ['mac', 'windows']

A list of plugin categories (typically operating systems) which the plugin will not operate on

find_aslr(*args)**find_requirements(context, config_path, requirement_root, requirement_type, shortcut=True)**

Determines if there is actually an unfulfilled *Requirement* waiting.

This ensures we do not carry out an expensive search when there is no need for a particular *Requirement*

Parameters

- **context** (*ContextInterface*) – Context on which to operate
- **config_path** (*str*) – Configuration path of the top-level requirement
- **requirement_root** (*RequirementInterface*) – Top-level requirement whose sub-requirements will all be searched
- **requirement_type** (*Union[Tuple[Type[RequirementInterface], ...], Type[RequirementInterface]*) – Type of requirement to find
- **shortcut** (*bool*) – Only returns requirements that live under unsatisfied requirements

Return type

List[Tuple[str, RequirementInterface]

Returns

A list of tuples containing the config_path, sub_config_path and requirement identifying the unsatisfied *Requirements*

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

List[RequirementInterface]

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration

- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

operating_system: *Optional*[*str*] = 'linux'

priority = 40

An ordering to indicate how soon this automagic should be run

symbol_class: *Optional*[*str*] =
'volatility3.framework.symbols.linux.LinuxKernelIntermedSymbols'

classmethod **unsatisfied**(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[*str*, *RequirementInterface*]

volatility3.framework.automagic.mac module

class **MacIntelStacker**

Bases: *StackerLayerInterface*

exclusion_list: *List*[*str*] = ['windows', 'linux']

The list operating systems/first-level plugin hierarchy that should exclude this stacker

classmethod **find_aslr**(*context*, *symbol_table*, *layer_name*, *compare_banner*="",
compare_banner_offset=0, *progress_callback*=None)

Determines the offset of the actual DTB in physical space and its symbol offset.

Return type

int

classmethod **stack**(*context*, *layer_name*, *progress_callback*=None)

Attempts to identify mac within this layer.

Return type

Optional[*DataLayerInterface*]

stack_order = 35

The order in which to attempt stacking, the lower the earlier

classmethod **stacker_slow_warning**()

classmethod `virtual_to_physical_address(addr)`

Converts a virtual mac address to a physical one (does not account of ASLR)

Return type

`int`

class `MacSymbolFinder(context, config_path)`

Bases: `SymbolFinder`

Mac symbol loader based on uname signature strings.

Basic initializer that allows configurables to access their own config settings.

banner_config_key: `str = 'kernel_banner'`

property `banners: Dict[bytes, List[str]]`

Creates a cached copy of the results, but only it's been requested.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property `config: HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property `config_path: str`

The configuration path on which this configurable lives.

property `context: ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

exclusion_list = `['windows', 'linux']`

A list of plugin categories (typically operating systems) which the plugin will not operate on

classmethod `find_aslr(context, symbol_table, layer_name, compare_banner="", compare_banner_offset=0, progress_callback=None)`

Determines the offset of the actual DTB in physical space and its symbol offset.

Return type

`int`

find_requirements(context, config_path, requirement_root, requirement_type, shortcut=True)

Determines if there is actually an unfulfilled *Requirement* waiting.

This ensures we do not carry out an expensive search when there is no need for a particular *Requirement*

Parameters

- **context** (`ContextInterface`) – Context on which to operate
- **config_path** (`str`) – Configuration path of the top-level requirement
- **requirement_root** (`RequirementInterface`) – Top-level requirement whose sub-requirements will all be searched

- **requirement_type** (`Union[Tuple[Type[RequirementInterface], ...], Type[RequirementInterface]]`) – Type of requirement to find
- **shortcut** (`bool`) – Only returns requirements that live under unsatisfied requirements

Return type

`List[Tuple[str, RequirementInterface]]`

Returns

A list of tuples containing the `config_path`, `sub_config_path` and requirement identifying the unsatisfied *Requirements*

classmethod `get_requirements()`

Returns a list of *RequirementInterface* objects required by this object.

Return type

`List[RequirementInterface]`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from `kwargs`.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

operating_system: `Optional[str] = 'mac'`

priority = 40

An ordering to indicate how soon this automagic should be run

symbol_class: `Optional[str] = 'volatility3.framework.symbols.mac.MacKernelIntermedSymbols'`

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return `[]`, it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

volatility3.framework.automagic.module module**class** `KernelModule(context, config_path, *args, **kwargs)`Bases: `AutomagicInterface`

Finds ModuleRequirements and ensures their layer, symbols and offsets

Basic initializer that allows configurables to access their own config settings.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type`HierarchicalDict`**property config:** `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

exclusion_list = []

A list of plugin categories (typically operating systems) which the plugin will not operate on

find_requirements(context, config_path, requirement_root, requirement_type, shortcut=True)Determines if there is actually an unfulfilled *Requirement* waiting.This ensures we do not carry out an expensive search when there is no need for a particular *Requirement***Parameters**

- **context** (`ContextInterface`) – Context on which to operate
- **config_path** (`str`) – Configuration path of the top-level requirement
- **requirement_root** (`RequirementInterface`) – Top-level requirement whose subrequirements will all be searched
- **requirement_type** (`Union[Tuple[Type[RequirementInterface], ...], Type[RequirementInterface]]`) – Type of requirement to find
- **shortcut** (`bool`) – Only returns requirements that live under unsatisfied requirements

Return type`List[Tuple[str, RequirementInterface]]`**Returns**A list of tuples containing the config_path, sub_config_path and requirement identifying the unsatisfied *Requirements***classmethod get_requirements()**

Returns a list of RequirementInterface objects required by this object.

Return type`List[RequirementInterface]`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

priority = 100

An ordering to indicate how soon this automagic should be run

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

volatility3.framework.automagic.pdbscan module

A module for scanning translation layers looking for Windows PDB records from loaded PE files.

This module contains a standalone scanner, and also a *ScannerInterface* based scanner for use within the framework by calling `scan()`.

class `KernelPDBScanner(context, config_path, *args, **kwargs)`

Bases: *AutomagicInterface*

Windows symbol loader based on PDB signatures.

An Automagic object that looks for all Intel translation layers and scans each of them for a pdb signature. When found, a search for a corresponding Intermediate Format data file is carried out and if found an appropriate symbol space is automatically loaded.

Once a specific kernel PDB signature has been found, a virtual address for the loaded kernel is determined by one of two methods. The first method assumes a specific mapping from the kernel's physical address to its virtual address (typically the kernel is loaded at its physical location plus a specific offset). The second method searches for a particular structure that lists the kernel module's virtual address, its size (not checked) and the module's name. This value is then used if one was not found using the previous method.

Basic initializer that allows configurables to access their own config settings.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

check_kernel_offset(context, vlayer, address, progress_callback=None)

Scans a virtual address.

Return type

Optional[Tuple[str, int, Dict[str, Union[bytes, str, int, None]]]]

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

determine_valid_kernel(context, potential_layers, progress_callback=None)

Runs through the identified potential kernels and verifies their suitability.

This carries out a scan using the pdb_signature scanner on a physical layer. It uses the results of the scan to determine the virtual offset of the kernel. On early windows implementations there is a fixed mapping between the physical and virtual addresses of the kernel. On more recent versions a search is conducted for a structure that will identify the kernel's virtual offset.

Parameters

- **context** (*ContextInterface*) – Context on which to operate
- **potential_layers** (*List[str]*) – List of layer names that the kernel might live at
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – Function taking a percentage and optional description to be called during expensive computations to indicate progress

Return type

Optional[Tuple[str, int, Dict[str, Union[bytes, str, int, None]]]]

Returns

A dictionary of valid kernels

exclusion_list = ['linux', 'mac']

A list of plugin categories (typically operating systems) which the plugin will not operate on

find_requirements(context, config_path, requirement_root, requirement_type, shortcut=True)

Determines if there is actually an unfulfilled *Requirement* waiting.

This ensures we do not carry out an expensive search when there is no need for a particular *Requirement*

Parameters

- **context** (*ContextInterface*) – Context on which to operate
- **config_path** (*str*) – Configuration path of the top-level requirement

- **requirement_root** (*RequirementInterface*) – Top-level requirement whose sub-requirements will all be searched
- **requirement_type** (*Union[Tuple[Type[RequirementInterface], ...], Type[RequirementInterface]*]) – Type of requirement to find
- **shortcut** (*bool*) – Only returns requirements that live under unsatisfied requirements

Return type

List[Tuple[str, RequirementInterface]

Returns

A list of tuples containing the *config_path*, *sub_config_path* and requirement identifying the unsatisfied *Requirements*

find_virtual_layers_from_req(*context, config_path, requirement*)

Traverses the requirement tree, rooted at *requirement* looking for virtual layers that might contain a windows PDB.

Returns a list of possible layers

Parameters

- **context** (*ContextInterface*) – The context in which the *requirement* lives
- **config_path** (*str*) – The path within the *context* for the *requirement*'s configuration variables
- **requirement** (*RequirementInterface*) – The root of the requirement tree to search for `:class:`~volatility3.framework.interfaces.layers.TranslationLayerRequirement`` objects to scan

Return type

List[str]

Returns

A list of (layer_name, scan_results)

get_physical_layer_name(*context, vlayer*)

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

List[RequirementInterface]

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from *kwargs*.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

max_pdb_size = 4194304

method_fixed_mapping(*context*, *vlayer*, *progress_callback=None*)

Return type

`Optional[Tuple[str, int, Dict[str, Union[bytes, str, int, None]]]]`

method_kdbg_offset(*context*, *vlayer*, *progress_callback=None*)

Return type

`Optional[Tuple[str, int, Dict[str, Union[bytes, str, int, None]]]]`

method_module_offset(*context*, *vlayer*, *progress_callback=None*)

Return type

`Optional[Tuple[str, int, Dict[str, Union[bytes, str, int, None]]]]`

method_slow_scan(*context*, *vlayer*, *progress_callback=None*)

Return type

`Optional[Tuple[str, int, Dict[str, Union[bytes, str, int, None]]]]`

methods = [<function KernelPDBScanner.method_kdbg_offset>, <function KernelPDBScanner.method_module_offset>, <function KernelPDBScanner.method_fixed_mapping>, <function KernelPDBScanner.method_slow_scan>]

priority = 30

An ordering to indicate how soon this automagic should be run

recurse_symbol_fulfiller(*context*, *valid_kernel*, *progress_callback=None*)

Fulfills the `SymbolTableRequirements` in `self._symbol_requirements` found by the `recurse_symbol_requirements`.

This pass will construct any requirements that may need it in the context it was passed

Parameters

- **context** (`ContextInterface`) – Context on which to operate
- **valid_kernel** (`Tuple[str, int, Dict[str, Union[bytes, str, int, None]]]`) – A list of offsets where valid kernels have been found
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – Means of providing the user with feedback during long processes

Return type

`None`

set_kernel_virtual_offset(*context*, *valid_kernel*)

Traverses the requirement tree, looking for `kernel_virtual_offset` values that may need setting and sets it based on the previously identified `valid_kernel`.

Parameters

- **context** (`ContextInterface`) – Context on which to operate and provide the kernel virtual offset
- **valid_kernel** (`Tuple[str, int, Dict[str, Union[bytes, str, int, None]]]`) – List of valid kernels and offsets

Return type

`None`

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

volatility3.framework.automagic.stacker module

This module attempts to automatically stack layers.

This automagic module fulfills `TranslationLayerRequirement` that are not already fulfilled, by attempting to stack as many layers on top of each other as possible. The base/lowest layer is derived from the “automagic.general.single_location” configuration path. Layers are then attempting in likely height order, and once a layer successfully stacks on top of the existing layers, it is removed from the possible choices list (so no layer type can exist twice in the layer stack).

class `LayerStacker(*args, **kwargs)`

Bases: `AutomagicInterface`

Builds up layers in a single stack.

This class mimics the volatility 2 style of stacking address spaces. It builds up various layers based on separate `StackerLayerInterface` classes. These classes are built up based on a `stack_order` class variable each has.

This has a high priority to provide other automagic modules as complete a context/configuration tree as possible. Upon completion it will re-call the `ConstructionMagic`, so that any stacked layers are actually constructed and added to the context.

Basic initializer that allows configurables to access their own config settings.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

create_stackers_list()

Creates the list of stackers to use based on the config option

Return type`List[Type[StackerLayerInterface]]`**exclusion_list** = []

A list of plugin categories (typically operating systems) which the plugin will not operate on

find_requirements(*context*, *config_path*, *requirement_root*, *requirement_type*, *shortcut=True*)

Determines if there is actually an unfulfilled *Requirement* waiting.

This ensures we do not carry out an expensive search when there is no need for a particular *Requirement*

Parameters

- **context** (*ContextInterface*) – Context on which to operate
- **config_path** (*str*) – Configuration path of the top-level requirement
- **requirement_root** (*RequirementInterface*) – Top-level requirement whose subrequirements will all be searched
- **requirement_type** (`Union[Tuple[Type[RequirementInterface], ...], Type[RequirementInterface]]`) – Type of requirement to find
- **shortcut** (*bool*) – Only returns requirements that live under unsatisfied requirements

Return type`List[Tuple[str, RequirementInterface]]`**Returns**

A list of tuples containing the *config_path*, *sub_config_path* and requirement identifying the unsatisfied *Requirements*

classmethod find_suitable_requirements(*context*, *config_path*, *requirement*, *stacked_layers*)

Looks for translation layer requirements and attempts to apply the stacked layers to it. If it succeeds it returns the configuration path and layer name where the stacked nodes were spliced into the tree.

Return type`Optional[Tuple[str, str]]`**Returns**

A tuple of a configuration path and layer name for the top of the stacked layers
or None if suitable requirements are not found

classmethod get_requirements()

Returns a list of *RequirementInterface* objects required by this object.

Return type`List[RequirementInterface]`**classmethod make_subconfig**(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from *kwargs*.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

priority = 10

An ordering to indicate how soon this automagic should be run

stack(context, config_path, requirement, progress_callback)

Stacks the various layers and attaches these to a specific requirement.

Parameters

- **context** (*ContextInterface*) – Context on which to operate
- **config_path** (`str`) – Configuration path under which to store stacking data
- **requirement** (*RequirementInterface*) – Requirement that should have layers stacked on it
- **progress_callback** (*Optional*[*Callable*[[`float`, `str`], `None`]]) – Function to provide callback progress

Return type

`None`

classmethod stack_layer(context, initial_layer, stack_set=None, progress_callback=None)

Stacks as many possible layers on top of the initial layer as can be done.

WARNING: This modifies the context provided and may pollute it with unnecessary layers Recommended use is to: 1. Pass in `context.clone()` instead of `context` 2. When provided the layer list, choose the desired layer 3. Build the configuration using `layer.build_configuration()` 4. Merge the configuration into the original context with `context.config.merge()` 5. Call Construction magic to reconstruct the layers from just the configuration

Parameters

- **context** (*ContextInterface*) – The context on which to operate
- **initial_layer** (`str`) – The name of the initial layer within the context
- **stack_set** (*List*[*Type*[*StackerLayerInterface*]]) – A list of *StackerLayerInterface* objects in the order they should be stacked
- **progress_callback** (*Optional*[*Callable*[[`float`, `str`], `None`]]) – A function to report progress during the process

Returns

A list of layer names that exist in the provided context, stacked in order (highest to lowest)

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[`str`, *RequirementInterface*]

choose_os_stackers(*plugin*)

Identifies the stackers that should be run, based on the plugin (and thus os) provided

Return type

`List[str]`

volatility3.framework.automagic.symbol_cache module**class** **CacheManagerInterface**(*filename*)

Bases: `VersionableInterface`

add_identifier(*location, operating_system, identifier*)

Adds an identifier to the store

find_location(*identifier, operating_system*)

Returns the location of the symbol file given the identifier

Parameters

- **identifier** (`bytes`) – string that uniquely identifies a particular symbol table
- **operating_system** (`Optional[str]`) – optional string to restrict identifiers to just those for a particular operating system

Return type

`Optional[str]`

Returns

The location of the symbols file that matches the identifier

get_hash(*location*)

Returns the hash of the JSON from within a location ISF

Return type

`Optional[str]`

get_identifier(*location*)

Returns an identifier based on a specific location or None

Return type

`Optional[bytes]`

get_identifier_dictionary(*operating_system=None, local_only=False*)

Returns a dictionary of identifiers and locations

Parameters

- **operating_system** (`Optional[str]`) – If set, limits responses to a specific operating system
- **local_only** (`bool`) – Returns only local locations

Return type

`Dict[bytes, str]`

Returns

A dictionary of identifiers mapped to a location

get_identifiers(*operating_system*)

Returns all identifiers for a particular operating system

Return type

`List[bytes]`

get_local_locations()

Returns a list of all the local locations

Return type

`Iterable[str]`

get_location_statistics(*location*)

Returns ISF statistics based on the location

Return type

`Optional[Tuple[int, int, int, int]]`

Returns

A tuple of base_types, types, enums, symbols, or None is location not found

update()

Locates all files under the symbol directories. Updates the cache with additions, modifications and removals. This also updates remote locations based on a cache timeout.

version = (0, 0, 0)

class IdentifierProcessor

Bases: `object`

abstract classmethod get_identifier(*json*)

Method to extract the identifier from a particular operating system's JSON

Return type

`Optional[bytes]`

Returns

identifier is valid or None if not found

operating_system = None

class LinuxIdentifier

Bases: `IdentifierProcessor`

classmethod get_identifier(*json*)

Method to extract the identifier from a particular operating system's JSON

Return type

`Optional[bytes]`

Returns

identifier is valid or None if not found

operating_system = 'linux'

class MacIdentifier

Bases: `IdentifierProcessor`

classmethod `get_identifier(json)`

Method to extract the identifier from a particular operating system's JSON

Return type

`Optional[bytes]`

Returns

identifier is valid or None if not found

`operating_system = 'mac'`

class `RemoteIdentifierFormat(location)`

Bases: `object`

process(*identifiers, operating_system*)

Return type

`Generator[Tuple[bytes, str], None, None]`

process_v1(*identifiers, operating_system*)

Return type

`Generator[Tuple[bytes, str], None, None]`

class `SqliteCache(filename)`

Bases: `CacheManagerInterface`

add_identifier(*location, operating_system, identifier*)

Adds an identifier to the store

find_location(*identifier, operating_system*)

Returns the location of the symbol file given the identifier. If multiple locations exist for an identifier, the last found is returned

Parameters

- **identifier** (`bytes`) – string that uniquely identifies a particular symbol table
- **operating_system** (`Optional[str]`) – optional string to restrict identifiers to just those for a particular operating system

Return type

`Optional[str]`

Returns

The location of the symbols file that matches the identifier or None

get_hash(*location*)

Returns the hash of the JSON from within a location ISF

Return type

`Optional[str]`

get_identifier(*location*)

Returns an identifier based on a specific location or None

Return type

`Optional[bytes]`

get_identifier_dictionary(*operating_system=None, local_only=False*)

Returns a dictionary of identifiers and locations

Parameters

- **operating_system** (*Optional[str]*) – If set, limits responses to a specific operating system
- **local_only** (*bool*) – Returns only local locations

Return type

Dict[bytes, str]

Returns

A dictionary of identifiers mapped to a location

get_identifiers(*operating_system*)

Returns all identifiers for a particular operating system

Return type

List[bytes]

get_local_locations()

Returns a list of all the local locations

Return type

Generator[str, None, None]

get_location_statistics(*location*)

Returns ISF statistics based on the location

Return type

Optional[Tuple[int, int, int, int]]

Returns

A tuple of base_types, types, enums, symbols, or None is location not found

is_url_local(*url*)

Determines whether an url is local or not

Return type

bool

update(*progress_callback=None*)

Locates all files under the symbol directories. Updates the cache with additions, modifications and removals. This also updates remote locations based on a cache timeout.

version = (1, 0, 0)

class SymbolCacheMagic(*args, **kwargs)

Bases: *AutomagicInterface*

Runs through all symbol tables and caches their identifiers

Basic initializer that allows configurables to access their own config settings.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

exclusion_list = []

A list of plugin categories (typically operating systems) which the plugin will not operate on

find_requirements(*context, config_path, requirement_root, requirement_type, shortcut=True*)

Determines if there is actually an unfulfilled *Requirement* waiting.

This ensures we do not carry out an expensive search when there is no need for a particular *Requirement*

Parameters

- **context** (*ContextInterface*) – Context on which to operate
- **config_path** (*str*) – Configuration path of the top-level requirement
- **requirement_root** (*RequirementInterface*) – Top-level requirement whose sub-requirements will all be searched
- **requirement_type** (*Union[Tuple[Type[RequirementInterface], ...], Type[RequirementInterface]*) – Type of requirement to find
- **shortcut** (*bool*) – Only returns requirements that live under unsatisfied requirements

Return type*List[Tuple[str, RequirementInterface]]***Returns**

A list of tuples containing the config_path, sub_config_path and requirement identifying the unsatisfied *Requirements*

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type*List[RequirementInterface]***classmethod make_subconfig**(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**priority = 0**

An ordering to indicate how soon this automagic should be run

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`**class** `WindowsIdentifier`

Bases: `IdentifierProcessor`

classmethod `generate(pdb_name, guid, age)`**Return type**`bytes`**classmethod** `get_identifier(json)`

Returns the identifier for the file if one can be found

Return type`Optional[bytes]`**operating_system** = 'windows'**separator** = '|'**volatility3.framework.automagic.symbol_finder module****class** `SymbolFinder(context, config_path)`

Bases: `AutomagicInterface`

Symbol loader based on signature strings.

Basic initializer that allows configurables to access their own config settings.

banner_config_key: `str` = 'banner'**property** `banners:` `Dict[bytes, List[str]]`

Creates a cached copy of the results, but only it's been requested.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

exclusion_list = []

A list of plugin categories (typically operating systems) which the plugin will not operate on

find_aslr: *Optional[Callable]* = None**find_requirements**(*context, config_path, requirement_root, requirement_type, shortcut=True*)

Determines if there is actually an unfulfilled *Requirement* waiting.

This ensures we do not carry out an expensive search when there is no need for a particular *Requirement*

Parameters

- **context** (*ContextInterface*) – Context on which to operate
- **config_path** (*str*) – Configuration path of the top-level requirement
- **requirement_root** (*RequirementInterface*) – Top-level requirement whose sub-requirements will all be searched
- **requirement_type** (*Union[Tuple[Type[RequirementInterface], ...], Type[RequirementInterface]*]) – Type of requirement to find
- **shortcut** (*bool*) – Only returns requirements that live under unsatisfied requirements

Return type*List[Tuple[str, RequirementInterface]]***Returns**

A list of tuples containing the config_path, sub_config_path and requirement identifying the unsatisfied *Requirements*

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type*List[RequirementInterface]***classmethod make_subconfig**(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**operating_system:** `Optional[str] = None`**priority = 40**

An ordering to indicate how soon this automagic should be run

symbol_class: `Optional[str] = None`**classmethod** `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`**volatility3.framework.automagic.windows module**

Module to identify the Directory Table Base and architecture of windows memory images.

This module contains a PageMapScanner that scans a physical layer to identify self-referential pointers. All windows versions include a self-referential pointer in their Directory Table Base's top table, in order to have a single offset that will allow manipulation of the page tables themselves.

In older windows version the self-referential pointer was at a specific fixed index within the table, which was different for each architecture. In very recent Windows versions, the self-referential pointer index has been randomized, so a different heuristic must be used. In these versions of windows it was found that the physical offset for the DTB was always within the range of 0x1a0000 to 0x1b0000. As such, a search for any self-referential pointer within these pages gives a high probability of being an accurate DTB.

The self-referential indices for older versions of windows are listed below:

Architecture	Index
x86	0x300
PAE	0x3
x64	0x1ED

class `DtbSelfRef32bit`

Bases: `DtbSelfReferential`

class `DtbSelfRef64bit`

Bases: `DtbSelfReferential`

class `DtbSelfRef64bitOldWindows`

Bases: `DtbSelfReferential`

class `DtbSelfRefPae`

Bases: `DtbSelfReferential`

class DtbSelfReferential(*layer_type*, *ptr_struct*, *mask*, *valid_range*, *reserved_bits*)

Bases: [object](#)

A generic DTB test which looks for a self-referential pointer at *any* index within the page.

class PageMapScanner(*tests*)

Bases: [ScannerInterface](#)

Scans through all pages using DTB tests to determine a dtb offset and architecture.

property context: [ContextInterface](#) | **None**

property layer_name: **str** | **None**

overlap = 16384

tests = [[<volatility3.framework.automagic.windows.DtbSelfRef64bit object>](#),
[<volatility3.framework.automagic.windows.DtbSelfRefPae object>](#),
[<volatility3.framework.automagic.windows.DtbSelfRef32bit object>](#)]

The default tests to run when searching for DTBs

thread_safe = **True**

version = (0, 0, 0)

class WinSwapLayers(*context*, *config_path*, **args*, ***kwargs*)

Bases: [AutomagicInterface](#)

Class to read swap_layers filenames from single-swap-layers, create the layers and populate the single-layers swap_layers.

Basic initializer that allows configurables to access their own config settings.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

[HierarchicalDict](#)

property config: [HierarchicalDict](#)

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: **str**

The configuration path on which this configurable lives.

property context: [ContextInterface](#)

The context object that this configurable belongs to/configuration is stored in.

exclusion_list = ['linux', 'mac']

A list of plugin categories (typically operating systems) which the plugin will not operate on

find_requirements(*context*, *config_path*, *requirement_root*, *requirement_type*, *shortcut=True*)

Determines if there is actually an unfulfilled *Requirement* waiting.

This ensures we do not carry out an expensive search when there is no need for a particular *Requirement*

Parameters

- **context** (*ContextInterface*) – Context on which to operate
- **config_path** (*str*) – Configuration path of the top-level requirement
- **requirement_root** (*RequirementInterface*) – Top-level requirement whose sub-requirements will all be searched
- **requirement_type** (*Union[Tuple[Type[RequirementInterface], ...], Type[RequirementInterface]*) – Type of requirement to find
- **shortcut** (*bool*) – Only returns requirements that live under unsatisfied requirements

Return type

List[Tuple[str, RequirementInterface]

Returns

A list of tuples containing the config_path, sub_config_path and requirement identifying the unsatisfied *Requirements*

static find_swap_requirement(*config, requirement*)

Takes a Translation layer and returns its swap_layer requirement.

Return type

Tuple[str, Optional[LayerListRequirement]]

classmethod get_requirements()

Returns the requirements of this plugin.

Return type

List[RequirementInterface]

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

priority = 10

An ordering to indicate how soon this automagic should be run

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`**class WindowsIntelStacker**Bases: `StackerLayerInterface`**exclusion_list:** `List[str] = ['mac', 'linux']`

The list operating systems/first-level plugin hierarchy that should exclude this stacker

classmethod stack(*context, layer_name, progress_callback=None*)

Attempts to determine and stack an intel layer on a physical layer where possible.

Where the DTB scan fails, it attempts a heuristic of checking for the DTB within a specific range. New versions of windows, with randomized self-referential pointers, appear to always load their dtb within a small specific range (`0x1a0000` and `0x1b0000`), so instead we scan for all self-referential pointers in that range, and ignore any that contain multiple self-references (since the DTB is very unlikely to point to itself more than once).

Return type`Optional[DataLayerInterface]`**stack_order** = 40

The order in which to attempt stacking, the lower the earlier

classmethod stacker_slow_warning()

```
test_sets = [('Detecting Self-referential pointer for recent windows',
[<volatility3.framework.automagic.windows.DtbSelfRef64bit object>], [(1376256,
1376256), (6619136, 655360)]), ('Older windows fixed location self-referential
pointers', [<volatility3.framework.automagic.windows.DtbSelfRefPae object>,
<volatility3.framework.automagic.windows.DtbSelfRef32bit object>,
<volatility3.framework.automagic.windows.DtbSelfRef64bitOldWindows object>],
[(196608, 16777216)])]
```

volatility3.framework.configuration package**Submodules****volatility3.framework.configuration.requirements module**

Contains standard Requirement types that all adhere to the RequirementInterface.

These requirement types allow plugins to request simple information types (such as strings, integers, etc) as well as indicating what they expect to be in the context (such as particular layers or symboltables).

class BooleanRequirement(*name, description=None, default=None, optional=False*)Bases: `SimpleTypeRequirement`

A requirement type that contains a boolean value.

Parameters

- **name** (`str`) – The name of the requirement
- **description** (`str`) – A short textual description of the requirement
- **default** (`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`) – The default value for the requirement if no value is provided

- **optional** (`bool`) – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Always raises a `TypeError` as instance requirements cannot have children.

config_value(*context*, *config_path*, *default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (`ContextInterface`) – the configuration store to find the value for this requirement
- **config_path** (`str`) – the configuration path of the instance of the requirement to be recovered
- **default** (`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`) – a default value to provide if the requirement's configuration value is not found

Return type

`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

property default: `int | bool | bytes | str | List[int | bool | bytes | str] | None`

Returns the default value if one is set.

property description: `str`

A short description of what the Requirement is designed to affect or achieve.

instance_type

alias of `bool`

property name: `str`

The name of the Requirement.

Names cannot contain `CONFIG_SEPARATOR` ('.' by default) since this is used within the configuration hierarchy.

property optional: `bool`

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Always raises a `TypeError` as instance requirements cannot have children.

property requirements: `Dict[str, RequirementInterface]`

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context*, *config_path*)

Validates the instance requirement based upon its *instance_type*.

Return type

`Dict[str, RequirementInterface]`

unsatisfied_children(*context*, *config_path*)

Method that will validate all child requirements.

Parameters

- **context** (`ContextInterface`) – the context containing the configuration data for this requirement
- **config_path** (`str`) – the configuration path of this instance of the requirement

Return type`Dict[str, RequirementInterface]`**Returns**

A dictionary of full configuration paths for each unsatisfied child-requirement

class BytesRequirement(*name, description=None, default=None, optional=False*)

Bases: `SimpleTypeRequirement`

A requirement type that contains a byte string.

Parameters

- **name** (`str`) – The name of the requirement
- **description** (`str`) – A short textual description of the requirement
- **default** (`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`) – The default value for the requirement if no value is provided
- **optional** (`bool`) – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Always raises a `TypeError` as instance requirements cannot have children.

config_value(*context, config_path, default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (`ContextInterface`) – the configuration store to find the value for this requirement
- **config_path** (`str`) – the configuration path of the instance of the requirement to be recovered
- **default** (`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`) – a default value to provide if the requirement's configuration value is not found

Return type`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

property default: `int | bool | bytes | str | List[int | bool | bytes | str] | None`

Returns the default value if one is set.

property description: `str`

A short description of what the Requirement is designed to affect or achieve.

instance_type

alias of `bytes`

property name: `str`

The name of the Requirement.

Names cannot contain `CONFIG_SEPARATOR` (‘.’ by default) since this is used within the configuration hierarchy.

property optional: `bool`

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Always raises a `TypeError` as instance requirements cannot have children.

property requirements: `Dict[str, RequirementInterface]`

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context*, *config_path*)

Validates the instance requirement based upon its *instance_type*.

Return type

`Dict[str, RequirementInterface]`

unsatisfied_children(*context*, *config_path*)

Method that will validate all child requirements.

Parameters

- **context** (`ContextInterface`) – the context containing the configuration data for this requirement
- **config_path** (`str`) – the configuration path of this instance of the requirement

Return type

`Dict[str, RequirementInterface]`

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

class ChoiceRequirement(*choices*, **args*, ***kwargs*)

Bases: `RequirementInterface`

Allows one from a choice of strings.

Constructs the object.

Parameters

choices (`List[str]`) – A list of possible string options that can be chosen from

add_requirement(*requirement*)

Adds a child to the list of requirements.

Parameters

requirement (`RequirementInterface`) – The requirement to add as a child-requirement

Return type

`None`

config_value(*context*, *config_path*, *default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (`ContextInterface`) – the configuration store to find the value for this requirement
- **config_path** (`str`) – the configuration path of the instance of the requirement to be recovered
- **default** (`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`) – a default value to provide if the requirement's configuration value is not found

Return type

`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

property default: `int | bool | bytes | str | List[int | bool | bytes | str] | None`

Returns the default value if one is set.

property description: `str`

A short description of what the Requirement is designed to affect or achieve.

property name: `str`

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

property optional: `bool`

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Removes a child from the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to remove as a child-requirement

Return type

`None`

property requirements: `Dict[str, RequirementInterface]`

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context, config_path*)

Validates the provided value to ensure it is one of the available choices.

Return type

`Dict[str, RequirementInterface]`

unsatisfied_children(*context, config_path*)

Method that will validate all child requirements.

Parameters

- **context** (*ContextInterface*) – the context containing the configuration data for this requirement
- **config_path** (`str`) – the configuration path of this instance of the requirement

Return type

`Dict[str, RequirementInterface]`

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

class ComplexListRequirement(*name, description=None, default=None, optional=False*)

Bases: *MultiRequirement, ConfigurableRequirementInterface*

Allows a variable length list of requirements.

Parameters

- **name** (`str`) – The name of the requirement
- **description** (`str`) – A short textual description of the requirement
- **default** (`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`) – The default value for the requirement if no value is provided
- **optional** (`bool`) – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Adds a child to the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to add as a child-requirement

Return type

None

build_configuration(*context, config_path, _*)

Proxies to a ConfigurableInterface if necessary.

Return type

HierarchicalDict

config_value(*context, config_path, default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (*ContextInterface*) – the configuration store to find the value for this requirement
- **config_path** (*str*) – the configuration path of the instance of the requirement to be recovered
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None*) – a default value to provide if the requirement's configuration value is not found

Return type

Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]

abstract construct(*context, config_path*)

Method for constructing within the context any required elements from subrequirements.

Return type

None

property default: *int | bool | bytes | str | List[int | bool | bytes | str] | None*

Returns the default value if one is set.

property description: *str*

A short description of what the Requirement is designed to affect or achieve.

classmethod get_requirements()

Return type

List[RequirementInterface]

property name: *str*

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

abstract new_requirement(*index*)

Builds a new requirement based on the specified index.

Return type

RequirementInterface

property optional: `bool`

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Removes a child from the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to remove as a child-requirement

Return type

`None`

property requirements: `Dict[str, RequirementInterface]`

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context, config_path*)

Validates the provided value to ensure it is one of the available choices.

Return type

`Dict[str, RequirementInterface]`

unsatisfied_children(*context, config_path*)

Method that will validate all child requirements.

Parameters

- **context** (*ContextInterface*) – the context containing the configuration data for this requirement
- **config_path** (`str`) – the configuration path of this instance of the requirement

Return type

`Dict[str, RequirementInterface]`

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

class IntRequirement(*name, description=None, default=None, optional=False*)

Bases: *SimpleTypeRequirement*

A requirement type that contains a single integer.

Parameters

- **name** (`str`) – The name of the requirement
- **description** (`str`) – A short textual description of the requirement
- **default** (`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`) – The default value for the requirement if no value is provided
- **optional** (`bool`) – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Always raises a `TypeError` as instance requirements cannot have children.

config_value(*context, config_path, default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (*ContextInterface*) – the configuration store to find the value for this requirement

- **config_path** (*str*) – the configuration path of the instance of the requirement to be re-covered
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None*) – a default value to provide if the requirement’s configuration value is not found

Return type*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]***property default:** *int | bool | bytes | str | List[int | bool | bytes | str] | None*

Returns the default value if one is set.

property description: *str*

A short description of what the Requirement is designed to affect or achieve.

instance_typealias of *int***property name:** *str*

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

property optional: *bool*

Whether the Requirement is optional or not.

remove_requirement(*requirement*)Always raises a *TypeError* as instance requirements cannot have children.**property requirements:** *Dict[str, RequirementInterface]*

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context, config_path*)Validates the instance requirement based upon its *instance_type*.**Return type***Dict[str, RequirementInterface]***unsatisfied_children**(*context, config_path*)

Method that will validate all child requirements.

Parameters

- **context** (*ContextInterface*) – the context containing the configuration data for this requirement
- **config_path** (*str*) – the configuration path of this instance of the requirement

Return type*Dict[str, RequirementInterface]***Returns**

A dictionary of full configuration paths for each unsatisfied child-requirement

class LayerListRequirement(*name, description=None, default=None, optional=False*)Bases: *ComplexListRequirement*

Allows a variable length list of layers that must exist.

Parameters

- **name** (*str*) – The name of the requirement

- **description** (*str*) – A short textual description of the requirement
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]*) – The default value for the requirement if no value is provided
- **optional** (*bool*) – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Adds a child to the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to add as a child-requirement

Return type

None

build_configuration(*context, config_path, _*)

Proxies to a ConfigurableInterface if necessary.

Return type

HierarchicalDict

config_value(*context, config_path, default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (*ContextInterface*) – the configuration store to find the value for this requirement
- **config_path** (*str*) – the configuration path of the instance of the requirement to be recovered
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]*) – a default value to provide if the requirement’s configuration value is not found

Return type

Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]

construct(*context, config_path*)

Method for constructing within the context any required elements from subrequirements.

Return type

None

property default: *int | bool | bytes | str | List[int | bool | bytes | str] | None*

Returns the default value if one is set.

property description: *str*

A short description of what the Requirement is designed to affect or achieve.

classmethod get_requirements()

Return type

List[RequirementInterface]

property name: *str*

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

new_requirement(*index*)

Constructs a new requirement based on the specified index.

Return type

RequirementInterface

property optional: `bool`

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Removes a child from the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to remove as a child-requirement

Return type

`None`

property requirements: `Dict[str, RequirementInterface]`

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context, config_path*)

Validates the provided value to ensure it is one of the available choices.

Return type

`Dict[str, RequirementInterface]`

unsatisfied_children(*context, config_path*)

Method that will validate all child requirements.

Parameters

- **context** (*ContextInterface*) – the context containing the configuration data for this requirement
- **config_path** (`str`) – the configuration path of this instance of the requirement

Return type

`Dict[str, RequirementInterface]`

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

class ListRequirement(*element_type=<class 'str'>, max_elements=0, min_elements=None, *args, **kwargs*)

Bases: *RequirementInterface*

Allows for a list of a specific type of requirement (all of which must be met for this requirement to be met) to be specified.

This roughly correlates to allowing a number of arguments to follow a command line parameter, such as a list of integers or a list of strings.

It is distinct from a multi-requirement which stores the subrequirements in a dictionary, not a list, and does not allow for a dynamic number of values.

Constructs the object.

Parameters

- **element_type** (`Type[Union[int, bool, bytes, str]]`) – The (requirement) type of each element within the list

- **contain** (*max_elements*; *The maximum number of acceptable elements this list can*) –
- **min_elements** (*Optional[int]*) – The minimum number of acceptable elements this list can contain

add_requirement(*requirement*)

Adds a child to the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to add as a child-requirement

Return type

None

config_value(*context, config_path, default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (*ContextInterface*) – the configuration store to find the value for this requirement
- **config_path** (*str*) – the configuration path of the instance of the requirement to be recovered
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None*) – a default value to provide if the requirement's configuration value is not found

Return type

Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]

property default: *int | bool | bytes | str | List[int | bool | bytes | str] | None*

Returns the default value if one is set.

property description: *str*

A short description of what the Requirement is designed to affect or achieve.

property name: *str*

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

property optional: *bool*

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Removes a child from the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to remove as a child-requirement

Return type

None

property requirements: *Dict[str, RequirementInterface]*

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context*, *config_path*)

Check the types on each of the returned values and their number and then call the element type's check for each one.

Return type

`Dict[str, RequirementInterface]`

unsatisfied_children(*context*, *config_path*)

Method that will validate all child requirements.

Parameters

- **context** (`ContextInterface`) – the context containing the configuration data for this requirement
- **config_path** (`str`) – the configuration path of this instance of the requirement

Return type

`Dict[str, RequirementInterface]`

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

class ModuleRequirement(*name*, *description=None*, *default=False*, *architectures=None*, *optional=False*)

Bases: `ConstructableRequirementInterface`, `ConfigurableRequirementInterface`

Parameters

- **name** (`str`) – The name of the requirement
- **description** (`str`) – A short textual description of the requirement
- **default** (`bool`) – The default value for the requirement if no value is provided
- **optional** (`bool`) – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Adds a child to the list of requirements.

Parameters

requirement (`RequirementInterface`) – The requirement to add as a child-requirement

Return type

`None`

build_configuration(*context*, *_*, *value*)

Builds the appropriate configuration for the specified requirement.

Return type

`HierarchicalDict`

config_value(*context*, *config_path*, *default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (`ContextInterface`) – the configuration store to find the value for this requirement
- **config_path** (`str`) – the configuration path of the instance of the requirement to be recovered
- **default** (`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`) – a default value to provide if the requirement's configuration value is not found

Return type

`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

construct(*context*, *config_path*)

Constructs the appropriate layer and adds it based on the class parameter.

Return type

`None`

property default: `int | bool | bytes | str | List[int | bool | bytes | str] | None`

Returns the default value if one is set.

property description: `str`

A short description of what the Requirement is designed to affect or achieve.

classmethod `get_requirements()`

Return type

`List[RequirementInterface]`

property name: `str`

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

property optional: `bool`

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Removes a child from the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to remove as a child-requirement

Return type

`None`

property requirements: `Dict[str, RequirementInterface]`

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context*, *config_path*)

Validate that the value is a valid module

Return type

`Dict[str, RequirementInterface]`

unsatisfied_children(*context*, *config_path*)

Method that will validate all child requirements.

Parameters

- **context** (*ContextInterface*) – the context containing the configuration data for this requirement
- **config_path** (`str`) – the configuration path of this instance of the requirement

Return type

`Dict[str, RequirementInterface]`

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

class MultiRequirement(*name, description=None, default=None, optional=False*)

Bases: [RequirementInterface](#)

Class to hold multiple requirements.

Technically the Interface could handle this, but it's an interface, so this is a concrete implementation.

Parameters

- **name** ([str](#)) – The name of the requirement
- **description** ([str](#)) – A short textual description of the requirement
- **default** ([Union\[int, bool, bytes, str, List\[Union\[int, bool, bytes, str\]\], None\]](#)) – The default value for the requirement if no value is provided
- **optional** ([bool](#)) – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Adds a child to the list of requirements.

Parameters

requirement ([RequirementInterface](#)) – The requirement to add as a child-requirement

Return type

[None](#)

config_value(*context, config_path, default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** ([ContextInterface](#)) – the configuration store to find the value for this requirement
- **config_path** ([str](#)) – the configuration path of the instance of the requirement to be recovered
- **default** ([Union\[int, bool, bytes, str, List\[Union\[int, bool, bytes, str\]\], None\]](#)) – a default value to provide if the requirement's configuration value is not found

Return type

[Union\[int, bool, bytes, str, List\[Union\[int, bool, bytes, str\]\], None\]](#)

property default: [int](#) | [bool](#) | [bytes](#) | [str](#) | [List\[int | bool | bytes | str\]](#) | [None](#)

Returns the default value if one is set.

property description: [str](#)

A short description of what the Requirement is designed to affect or achieve.

property name: [str](#)

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

property optional: [bool](#)

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Removes a child from the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to remove as a child-requirement

Return type

None

property requirements: *Dict[str, RequirementInterface]*

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context, config_path*)

Method to validate the value stored at *config_path* for the configuration object against a context.

Returns a list containing its own name (or multiple unsatisfied requirement names) when invalid

Parameters

- **context** (*ContextInterface*) – The context object containing the configuration for this requirement
- **config_path** (*str*) – The configuration path for this requirement to test satisfaction

Return type

Dict[str, RequirementInterface]

Returns

A dictionary of configuration-paths to requirements that could not be satisfied

unsatisfied_children(*context, config_path*)

Method that will validate all child requirements.

Parameters

- **context** (*ContextInterface*) – the context containing the configuration data for this requirement
- **config_path** (*str*) – the configuration path of this instance of the requirement

Return type

Dict[str, RequirementInterface]

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

class PluginRequirement(*name, description=None, default=False, optional=False, plugin=None, version=None*)

Bases: *VersionRequirement*

Parameters

- **name** (*str*) – The name of the requirement
- **description** (*str*) – A short textual description of the requirement
- **default** (*bool*) – The default value for the requirement if no value is provided
- **optional** (*bool*) – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Adds a child to the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to add as a child-requirement

Return type

None

config_value(*context*, *config_path*, *default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (*ContextInterface*) – the configuration store to find the value for this requirement
- **config_path** (*str*) – the configuration path of the instance of the requirement to be recovered
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None*) – a default value to provide if the requirement's configuration value is not found

Return type

Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]

property default: *int | bool | bytes | str | List[int | bool | bytes | str] | None*

Returns the default value if one is set.

property description: *str*

A short description of what the Requirement is designed to affect or achieve.

classmethod matches_required(*required*, *version*)

Return type

bool

property name: *str*

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

property optional: *bool*

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Removes a child from the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to remove as a child-requirement

Return type

None

property requirements: *Dict[str, RequirementInterface]*

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context*, *config_path*)

Method to validate the value stored at *config_path* for the configuration object against a context.

Returns a list containing its own name (or multiple unsatisfied requirement names) when invalid

Parameters

- **context** (*ContextInterface*) – The context object containing the configuration for this requirement
- **config_path** (*str*) – The configuration path for this requirement to test satisfaction

Return type

Dict[str, RequirementInterface]

Returns

A dictionary of configuration-paths to requirements that could not be satisfied

unsatisfied_children(*context*, *config_path*)

Method that will validate all child requirements.

Parameters

- **context** (*ContextInterface*) – the context containing the configuration data for this requirement
- **config_path** (*str*) – the configuration path of this instance of the requirement

Return type

Dict[str, RequirementInterface]

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

class StringRequirement(*name*, *description=None*, *default=None*, *optional=False*)

Bases: *SimpleTypeRequirement*

A requirement type that contains a single unicode string.

Parameters

- **name** (*str*) – The name of the requirement
- **description** (*str*) – A short textual description of the requirement
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]*) – The default value for the requirement if no value is provided
- **optional** (*bool*) – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Always raises a *TypeError* as instance requirements cannot have children.

config_value(*context*, *config_path*, *default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (*ContextInterface*) – the configuration store to find the value for this requirement
- **config_path** (*str*) – the configuration path of the instance of the requirement to be recovered

- **default** (`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None`)
– a default value to provide if the requirement’s configuration value is not found

Return type

`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

property default: `int | bool | bytes | str | List[int | bool | bytes | str] | None`

Returns the default value if one is set.

property description: `str`

A short description of what the Requirement is designed to affect or achieve.

instance_type

alias of `str`

property name: `str`

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

property optional: `bool`

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Always raises a `TypeError` as instance requirements cannot have children.

property requirements: `Dict[str, RequirementInterface]`

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context, config_path*)

Validates the instance requirement based upon its *instance_type*.

Return type

`Dict[str, RequirementInterface]`

unsatisfied_children(*context, config_path*)

Method that will validate all child requirements.

Parameters

- **context** (`ContextInterface`) – the context containing the configuration data for this requirement
- **config_path** (`str`) – the configuration path of this instance of the requirement

Return type

`Dict[str, RequirementInterface]`

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

class SymbolTableRequirement(*args, **kwargs)

Bases: `ConstructableRequirementInterface, ConfigurableRequirementInterface`

Class maintaining the limitations on what sort of symbol spaces are acceptable.

Parameters

- **name** – The name of the requirement
- **description** – A short textual description of the requirement

- **default** – The default value for the requirement if no value is provided
- **optional** – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Adds a child to the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to add as a child-requirement

Return type

None

build_configuration(*context*, *_*, *value*)

Builds the appropriate configuration for the specified requirement.

Return type

HierarchicalDict

config_value(*context*, *config_path*, *default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (*ContextInterface*) – the configuration store to find the value for this requirement
- **config_path** (*str*) – the configuration path of the instance of the requirement to be recovered
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None*) – a default value to provide if the requirement’s configuration value is not found

Return type

Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]

construct(*context*, *config_path*)

Constructs the symbol space within the context based on the subrequirements.

Return type

None

property default: *int* | *bool* | *bytes* | *str* | *List[int | bool | bytes | str]* | *None*

Returns the default value if one is set.

property description: *str*

A short description of what the Requirement is designed to affect or achieve.

property name: *str*

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

property optional: *bool*

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Removes a child from the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to remove as a child-requirement

Return type`None`**property requirements:** `Dict[str, RequirementInterface]`

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context*, *config_path*)

Validate that the value is a valid within the symbol space of the provided context.

Return type`Dict[str, RequirementInterface]`**unsatisfied_children**(*context*, *config_path*)

Method that will validate all child requirements.

Parameters

- **context** (`ContextInterface`) – the context containing the configuration data for this requirement
- **config_path** (`str`) – the configuration path of this instance of the requirement

Return type`Dict[str, RequirementInterface]`**Returns**

A dictionary of full configuration paths for each unsatisfied child-requirement

class TranslationLayerRequirement(*name*, *description=None*, *default=None*, *optional=False*, *oses=None*, *architectures=None*)

Bases: `ConstructableRequirementInterface`, `ConfigurableRequirementInterface`

Class maintaining the limitations on what sort of translation layers are acceptable.

Constructs a Translation Layer Requirement.

The configuration option's value will be the name of the layer once it exists in the store

Parameters

- **name** (`str`) – Name of the configuration requirement
- **description** (`str`) – Description of the configuration requirement
- **default** (`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`) – A default value (should not be used for TranslationLayers)
- **optional** (`bool`) – Whether the translation layer is required or not
- **oses** (`List`) – A list of valid operating systems which can satisfy this requirement
- **architectures** (`List`) – A list of valid architectures which can satisfy this requirement

add_requirement(*requirement*)

Adds a child to the list of requirements.

Parameters

requirement (`RequirementInterface`) – The requirement to add as a child-requirement

Return type`None`

build_configuration(*context*, *_*, *value*)

Builds the appropriate configuration for the specified requirement.

Return type

HierarchicalDict

config_value(*context*, *config_path*, *default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (*ContextInterface*) – the configuration store to find the value for this requirement
- **config_path** (*str*) – the configuration path of the instance of the requirement to be recovered
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None*) – a default value to provide if the requirement’s configuration value is not found

Return type

Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]

construct(*context*, *config_path*)

Constructs the appropriate layer and adds it based on the class parameter.

Return type

None

property default: *int | bool | bytes | str | List[int | bool | bytes | str] | None*

Returns the default value if one is set.

property description: *str*

A short description of what the Requirement is designed to affect or achieve.

property name: *str*

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

property optional: *bool*

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Removes a child from the list of requirements.

Parameters

- **requirement** (*RequirementInterface*) – The requirement to remove as a child-requirement

Return type

None

property requirements: *Dict[str, RequirementInterface]*

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context*, *config_path*)

Validate that the value is a valid layer name and that the layer adheres to the requirements.

Return type

Dict[str, RequirementInterface]

unsatisfied_children(*context*, *config_path*)

Method that will validate all child requirements.

Parameters

- **context** (*ContextInterface*) – the context containing the configuration data for this requirement
- **config_path** (*str*) – the configuration path of this instance of the requirement

Return type

Dict[str, RequirementInterface]

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

class URIRequirement(*name*, *description=None*, *default=None*, *optional=False*)

Bases: *StringRequirement*

A requirement type that contains a single unicode string that is a valid URI.

Parameters

- **name** (*str*) – The name of the requirement
- **description** (*str*) – A short textual description of the requirement
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]*) – The default value for the requirement if no value is provided
- **optional** (*bool*) – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Always raises a *TypeError* as instance requirements cannot have children.

config_value(*context*, *config_path*, *default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (*ContextInterface*) – the configuration store to find the value for this requirement
- **config_path** (*str*) – the configuration path of the instance of the requirement to be recovered
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]*) – a default value to provide if the requirement's configuration value is not found

Return type

Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]

property default: *int | bool | bytes | str | List[int | bool | bytes | str] | None*

Returns the default value if one is set.

property description: *str*

A short description of what the Requirement is designed to affect or achieve.

instance_type

alias of *str*

classmethod `location_from_file(filename)`

Returns the URL location from a file parameter (which may be a URL)

Parameters

filename (`str`) – The path to the file (either an absolute, relative, or URL path)

Return type

`str`

Returns

The URL for the location of the file

property name: `str`

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

property optional: `bool`

Whether the Requirement is optional or not.

remove_requirement(requirement)

Always raises a `TypeError` as instance requirements cannot have children.

property requirements: `Dict[str, RequirementInterface]`

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(context, config_path)

Validates the instance requirement based upon its *instance_type*.

Return type

`Dict[str, RequirementInterface]`

unsatisfied_children(context, config_path)

Method that will validate all child requirements.

Parameters

- **context** (`ContextInterface`) – the context containing the configuration data for this requirement
- **config_path** (`str`) – the configuration path of this instance of the requirement

Return type

`Dict[str, RequirementInterface]`

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

class `VersionRequirement(name, description=None, default=False, optional=False, component=None, version=None)`

Bases: `RequirementInterface`

Parameters

- **name** (`str`) – The name of the requirement
- **description** (`str`) – A short textual description of the requirement
- **default** (`bool`) – The default value for the requirement if no value is provided
- **optional** (`bool`) – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Adds a child to the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to add as a child-requirement

Return type

None

config_value(*context, config_path, default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (*ContextInterface*) – the configuration store to find the value for this requirement
- **config_path** (*str*) – the configuration path of the instance of the requirement to be recovered
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None*) – a default value to provide if the requirement's configuration value is not found

Return type

Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]

property default: *int | bool | bytes | str | List[int | bool | bytes | str] | None*

Returns the default value if one is set.

property description: *str*

A short description of what the Requirement is designed to affect or achieve.

classmethod matches_required(*required, version*)

Return type

bool

property name: *str*

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

property optional: *bool*

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Removes a child from the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to remove as a child-requirement

Return type

None

property requirements: *Dict[str, RequirementInterface]*

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context*, *config_path*)

Method to validate the value stored at *config_path* for the configuration object against a context.

Returns a list containing its own name (or multiple unsatisfied requirement names) when invalid

Parameters

- **context** (*ContextInterface*) – The context object containing the configuration for this requirement
- **config_path** (*str*) – The configuration path for this requirement to test satisfaction

Return type

Dict[str, RequirementInterface]

Returns

A dictionary of configuration-paths to requirements that could not be satisfied

unsatisfied_children(*context*, *config_path*)

Method that will validate all child requirements.

Parameters

- **context** (*ContextInterface*) – the context containing the configuration data for this requirement
- **config_path** (*str*) – the configuration path of this instance of the requirement

Return type

Dict[str, RequirementInterface]

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

volatility3.framework.constants package

Volatility 3 Constants.

Stores all the constant values that are generally fixed throughout volatility This includes default scanning block sizes, etc.

AUTOMAGIC_CONFIG_PATH = 'automagic'

The root section within the context configuration for automagic values

BANG = '!'

Constant used to delimit table names from type names when referring to a symbol

CACHE_PATH = '/home/docs/.cache/volatility3'

Default path to store cached data

CACHE_SQLITE_SCHEMA_VERSION = 1

Version for the sqlite3 cache schema

IDENTIFIERS_FILENAME = 'identifier.cache'

Default location to record information about available identifiers

ISF_EXTENSIONS = ['.json', '.json.xz', '.json.gz', '.json.bz2']

List of accepted extensions for ISF files

ISF_MINIMUM_DEPRECATED = (3, 9, 9)

The highest version of the ISF that's deprecated (usually higher than supported)

ISF_MINIMUM_SUPPORTED = (2, 0, 0)

The minimum supported version of the Intermediate Symbol Format

LOGLEVEL_V = 9

Logging level for a single -v

LOGLEVEL_VV = 8

Logging level for -vv

LOGLEVEL_VVV = 7

Logging level for -vvv

LOGLEVEL_VVVV = 6

Logging level for -vvvv

OFFLINE = False

Whether to go online to retrieve missing/necessary JSON files

PACKAGE_VERSION = '2.5.2'

The canonical version of the volatility3 package

PARALLELISM = Parallelism.Off

Default value to the parallelism setting used throughout volatility

**PLUGINS_PATH = ['/home/docs/checkouts/readthedocs.org/user_builds/volatility3/checkouts/stable/volatility3/plugins',
'/home/docs/checkouts/readthedocs.org/user_builds/volatility3/checkouts/stable/volatility3/framework/plugins']**

Default list of paths to load plugins from (volatility3/plugins and volatility3/framework/plugins)

class Parallelism(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: [IntEnum](#)

An enumeration listing the different types of parallelism applied to volatility.

Multiprocessing = 2

Off = 0

Threading = 1

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(length=1, byteorder='big', *, signed=False)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An OverflowError is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

ProgressCallback

Type information for ProgressCallback objects

alias of `Optional[Callable[[float, str], None]]`

REMOTE_ISF_URL = None

Remote URL to query for a list of ISF addresses

SQLITE_CACHE_PERIOD = '-3 days'

SQLite time modifier for how long each item is valid in the cache for

```
SYMBOL_BASEPATHS = ['/home/docs/checkouts/readthedocs.org/user_builds/volatility3/  
checkouts/stable/volatility3/symbols',  
'/home/docs/checkouts/readthedocs.org/user_builds/volatility3/checkouts/stable/  
volatility3/framework/symbols']
```

Default list of paths to load symbols from (volatility3/symbols and volatility3/framework/symbols)

Subpackages

volatility3.framework.constants.linux package

Volatility 3 Linux Constants.

Linux-specific values that aren't found in debug symbols

PAGE_SHIFT = 12

The value hard coded from the Linux Kernel (hence not extracted from the layer itself)

volatility3.framework.constants.windows package

Volatility 3 Windows Constants.

Windows-specific values that aren't found in debug symbols

```
KERNEL_MODULE_NAMES = ['ntkrnlmp', 'ntkrnlpa', 'ntkrpamp', 'ntoskrnl']
```

The list of names that kernel modules can have within the windows OS

volatility3.framework.contexts package

A *Context* maintains the accumulated state required for various plugins and framework functions.

This has been made an object to allow quick swapping and changing of contexts, to allow a plugin to act on multiple different contexts without them interfering with each other.

class ConfigurableModule(*context, config_path, name*)

Bases: *Module, ConfigurableInterface*

Constructs a new os-independent module.

Parameters

- **context** (*ContextInterface*) – The context within which this module will exist
- **config_path** (*str*) – The path within the context’s configuration tree
- **name** (*str*) – The name of the module

build_configuration()

Builds the configuration dictionary for this specific Module

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

Context that the module uses.

classmethod create(*context, module_name, layer_name, offset, **kwargs*)

Return type

Module

get_absolute_symbol_address(*name*)

Returns the absolute address of the symbol within this module

Return type

int

get_enumeration(*name*)

Returns an enumeration from the module’s symbol table.

Return type

Template

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

List[RequirementInterface]

get_symbol(*name*)

Returns a symbol object from the module’s symbol table.

Return type*SymbolInterface***get_symbols_by_absolute_location**(*offset*, *size=0*)

Returns the symbols within this module that live at the specified absolute offset provided.

Return type*List[str]***get_type**(*name*)

Returns a type from the module's symbol table.

Return type*Template***has_enumeration**(*name*)

Determines whether an enumeration is present in the module's symbol table.

Return type*bool***has_symbol**(*name*)

Determines whether a symbol is present in the module's symbol table.

Return type*bool***has_type**(*name*)

Determines whether a type is present in the module's symbol table.

Return type*bool***property layer_name:** *str*

Layer name in which the Module resides.

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property name:** *str*

The name of the constructed module.

object(*object_type*, *offset=None*, *native_layer_name=None*, *absolute=False*, ***kwargs*)

Returns an object created using the `symbol_table_name` and `layer_name` of the Module.

Parameters

- **object_type** (*str*) – Name of the type/enumeration (within the module) to construct
- **offset** (*int*) – The location of the object, ignored when *symbol_type* is *SYMBOL*
- **native_layer_name** (*Optional[str]*) – Name of the layer in which constructed objects are made (for pointers)
- **absolute** (*bool*) – whether the type’s offset is absolute within memory or relative to the module

Return type*ObjectInterface*

object_from_symbol(*symbol_name*, *native_layer_name=None*, *absolute=False*, *object_type=None*, ***kwargs*)

Returns an object based on a specific symbol (containing type and offset information) and the *layer_name* of the Module. This will throw a *ValueError* if the symbol does not contain an associated type, or if the symbol name is invalid. It will throw a *SymbolError* if the symbol cannot be found.

Parameters

- **symbol_name** (*str*) – Name of the symbol (within the module) to construct
- **native_layer_name** (*Optional[str]*) – Name of the layer in which constructed objects are made (for pointers)
- **absolute** (*bool*) – whether the symbol’s address is absolute or relative to the module
- **object_type** (*Union[str, ObjectInterface, None]*) – Override for the type from the symbol to use (or if the symbol type is missing)

Return type*ObjectInterface*

property offset: *int*

Returns the offset that the module resides within the layer of *layer_name*.

property symbol_table_name: *str*

The name of the symbol table associated with this module

property symbols

Lists the symbols contained in the symbol table for this module

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type*Dict[str, RequirementInterface]*

class Context

Bases: *ContextInterface*

Maintains the context within which to construct objects.

The context object is the main method of carrying around state that's been constructed for the purposes of investigating memory. It contains a `symbol_space` of all the symbols that can be accessed by plugins using the context. It also contains the memory made up of data and translation layers, and it contains a factory method for creating new objects.

Other context objects can be constructed as long as they support the `ContextInterface`. This is the primary context object to be used in the volatility framework. It maintains the

Initializes the context.

add_layer(*layer*)

Adds a named translation layer to the context.

Parameters

layer (`DataLayerInterface`) – The layer to be added to the memory

Raises

`volatility3.framework.exceptions.LayerException` – if the layer is already present, or has unmet dependencies

Return type

`None`

add_module(*module*)

Adds a named module to the context.

Parameters

module (`ModuleInterface`) – The module to be added to the module object collection

Raises

`volatility3.framework.exceptions.VolatilityException` – if the module is already present, or has unmet dependencies

clone()

Produce a clone of the context (and configuration), allowing modifications to be made without affecting any mutable objects in the original.

Memory constraints may become an issue for this function depending on how much is actually stored in the context

Return type

`ContextInterface`

property config: `HierarchicalDict`

Returns a mutable copy of the configuration, but does not allow the whole configuration to be altered.

property layers: `LayerContainer`

A LayerContainer object, allowing access to all data and translation layers currently available within the context.

module(*module_name*, *layer_name*, *offset*, *native_layer_name*=None, *size*=None)

Constructs a new os-independent module.

Parameters

- **module_name** (`str`) – The name of the module
- **layer_name** (`str`) – The layer within the context in which the module exists
- **offset** (`int`) – The offset at which the module exists in the layer
- **native_layer_name** (`Optional[str]`) – The default native layer for objects constructed by the module

- **size** (`Optional[int]`) – The size, in bytes, that the module occupies from offset location within the layer named `layer_name`

Return type*ModuleInterface***property modules:** *ModuleContainer*

A container for modules loaded in this context

object(*object_type, layer_name, offset, native_layer_name=None, **arguments*)

Object factory, takes a context, symbol, offset and optional layername.

Looks up the layername in the context, finds the object template based on the symbol, and constructs an object using the object template on the layer at the offset.

Parameters

- **object_type** (`Union[str, Template]`) – The name (or template) of the symbol type on which to construct the object. If this is a name, it should contain an explicit table name.
- **layer_name** (`str`) – The name of the layer on which to construct the object
- **offset** (`int`) – The offset within the layer at which the data used to create the object lives
- **native_layer_name** (`Optional[str]`) – The name of the layer the object references (for pointers) if different to `layer_name`

Return type*ObjectInterface***Returns**

A fully constructed object

property symbol_space: *SymbolSpaceInterface*

The space of all symbols that can be accessed within this context.

class Module(*context, config_path, name*)Bases: *ModuleInterface*

Constructs a new os-independent module.

Parameters

- **context** (*ContextInterface*) – The context within which this module will exist
- **config_path** (`str`) – The path within the context's configuration tree
- **name** (`str`) – The name of the module

build_configuration()

Builds the configuration dictionary for this specific Module

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: *ContextInterface*

Context that the module uses.

classmethod `create(context, module_name, layer_name, offset, **kwargs)`

Return type

Module

get_absolute_symbol_address(name)

Returns the absolute address of the symbol within this module

Return type

int

get_enumeration(name)

Returns an enumeration from the module's symbol table.

Return type

Template

classmethod `get_requirements()`

Returns a list of RequirementInterface objects required by this object.

Return type

List[RequirementInterface]

get_symbol(name)

Returns a symbol object from the module's symbol table.

Return type

SymbolInterface

get_symbols_by_absolute_location(offset, size=0)

Returns the symbols within this module that live at the specified absolute offset provided.

Return type

List[str]

get_type(name)

Returns a type from the module's symbol table.

Return type

Template

has_enumeration(name)

Determines whether an enumeration is present in the module's symbol table.

Return type

bool

has_symbol(name)

Determines whether a symbol is present in the module's symbol table.

Return type

bool

has_type(name)

Determines whether a type is present in the module's symbol table.

Return type

bool

property `layer_name: str`

Layer name in which the Module resides.

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property name: *str*

The name of the constructed module.

object(*object_type, offset=None, native_layer_name=None, absolute=False, **kwargs*)

Returns an object created using the symbol_table_name and layer_name of the Module.

Parameters

- **object_type** (*str*) – Name of the type/enumeration (within the module) to construct
- **offset** (*int*) – The location of the object, ignored when symbol_type is SYMBOL
- **native_layer_name** (*Optional[str]*) – Name of the layer in which constructed objects are made (for pointers)
- **absolute** (*bool*) – whether the type's offset is absolute within memory or relative to the module

Return type

ObjectInterface

object_from_symbol(*symbol_name, native_layer_name=None, absolute=False, object_type=None, **kwargs*)

Returns an object based on a specific symbol (containing type and offset information) and the layer_name of the Module. This will throw a ValueError if the symbol does not contain an associated type, or if the symbol name is invalid. It will throw a SymbolError if the symbol cannot be found.

Parameters

- **symbol_name** (*str*) – Name of the symbol (within the module) to construct
- **native_layer_name** (*Optional[str]*) – Name of the layer in which constructed objects are made (for pointers)
- **absolute** (*bool*) – whether the symbol's address is absolute or relative to the module
- **object_type** (*Union[str, ObjectInterface, None]*) – Override for the type from the symbol to use (or if the symbol type is missing)

Return type

ObjectInterface

property offset: *int*

Returns the offset that the module resides within the layer of layer_name.

property `symbol_table_name`: `str`

The name of the symbol table associated with this module

property `symbols`

Lists the symbols contained in the symbol table for this module

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

class `ModuleCollection(modules=None)`

Bases: `ModuleContainer`

Class to contain a collection of SizedModules and reason about their contents.

add_module(*module*)

Adds a module to the module collection

This will throw an exception if the required dependencies are not met

Parameters

module (`ModuleInterface`) – the module to add to the list of modules (based on module.name)

Return type

`None`

deduplicate()

Returns a new deduplicated ModuleCollection featuring no repeated modules (based on data hash)

All 0 sized modules will have identical hashes and are therefore included in the deduplicated version

Return type

`ModuleCollection`

free_module_name(*prefix='module'*)

Returns an unused module name

Return type

`str`

get(*k*, *d*) → *D*[*k*] if *k* in *D*, else *d*. *d* defaults to None.

get_module_symbols_by_absolute_location(*offset, size=0*)

Returns a tuple of (module_name, list_of_symbol_names) for each module, where symbols live at the absolute offset in memory provided.

Return type

`Iterable[Tuple[str, List[str]]]`

get_modules_by_symbol_tables(*symbol_table*)

Returns the modules which use the specified symbol table name

Return type

`Iterable[str]`

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

property modules: `ModuleCollection`

A name indexed dictionary of modules using that name in this collection.

values() → an object providing a view on D's values

class SizedModule(*context, config_path, name*)

Bases: `Module`

Constructs a new os-independent module.

Parameters

- **context** (`ContextInterface`) – The context within which this module will exist
- **config_path** (`str`) – The path within the context's configuration tree
- **name** (`str`) – The name of the module

build_configuration()

Builds the configuration dictionary for this specific Module

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

Context that the module uses.

classmethod create(*context, module_name, layer_name, offset, **kwargs*)

Return type

`Module`

get_absolute_symbol_address(*name*)

Returns the absolute address of the symbol within this module

Return type

`int`

get_enumeration(*name*)

Returns an enumeration from the module's symbol table.

Return type

`Template`

classmethod `get_requirements()`

Returns a list of RequirementInterface objects required by this object.

Return type

`List[RequirementInterface]`

get_symbol(*name*)

Returns a symbol object from the module's symbol table.

Return type

`SymbolInterface`

get_symbols_by_absolute_location(*offset*, *size=0*)

Returns the symbols within this module that live at the specified absolute offset provided.

Return type

`List[str]`

get_type(*name*)

Returns a type from the module's symbol table.

Return type

`Template`

has_enumeration(*name*)

Determines whether an enumeration is present in the module's symbol table.

Return type

`bool`

has_symbol(*name*)

Determines whether a symbol is present in the module's symbol table.

Return type

`bool`

has_type(*name*)

Determines whether a type is present in the module's symbol table.

Return type

`bool`

property `hash: str`

Hashes the module for equality checks.

The mapping should be sorted and should be quicker than reading the data We turn it into JSON to make a common string and use a quick hash, because collisions are unlikely

property `layer_name: str`

Layer name in which the Module resides.

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration

- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property name: `str`

The name of the constructed module.

object(*object_type*, *offset=None*, *native_layer_name=None*, *absolute=False*, ***kwargs*)

Returns an object created using the `symbol_table_name` and `layer_name` of the Module.

Parameters

- **object_type** (`str`) – Name of the type/enumeration (within the module) to construct
- **offset** (`int`) – The location of the object, ignored when `symbol_type` is `SYMBOL`
- **native_layer_name** (`Optional[str]`) – Name of the layer in which constructed objects are made (for pointers)
- **absolute** (`bool`) – whether the type's offset is absolute within memory or relative to the module

Return type

`ObjectInterface`

object_from_symbol(*symbol_name*, *native_layer_name=None*, *absolute=False*, *object_type=None*, ***kwargs*)

Returns an object based on a specific symbol (containing type and offset information) and the `layer_name` of the Module. This will throw a `ValueError` if the symbol does not contain an associated type, or if the symbol name is invalid. It will throw a `SymbolError` if the symbol cannot be found.

Parameters

- **symbol_name** (`str`) – Name of the symbol (within the module) to construct
- **native_layer_name** (`Optional[str]`) – Name of the layer in which constructed objects are made (for pointers)
- **absolute** (`bool`) – whether the symbol's address is absolute or relative to the module
- **object_type** (`Union[str, ObjectInterface, None]`) – Override for the type from the symbol to use (or if the symbol type is missing)

Return type

`ObjectInterface`

property offset: `int`

Returns the offset that the module resides within the layer of `layer_name`.

property size: `int`

Returns the size of the module (0 for unknown size)

property symbol_table_name: `str`

The name of the symbol table associated with this module

property symbols

Lists the symbols contained in the symbol table for this module

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

get_module_wrapper(*method*)

Returns a symbol using the `symbol_table_name` of the Module.

Return type

`Callable`

volatility3.framework.interfaces package

The interfaces module contains the API interface for the core volatility framework.

These interfaces should help developers attempting to write components for the main framework and help them understand how to use the internal components of volatility to write plugins.

Submodules

volatility3.framework.interfaces.automagic module

Defines the automagic interfaces for populating the context before a plugin runs.

Automagic objects attempt to automatically fill configuration values that a user has not filled.

class `AutomagicInterface(context, config_path, *args, **kwargs)`

Bases: `ConfigurableInterface`

Class that defines an automagic component that can help fulfill *Requirements*

These classes are callable with the following parameters:

Parameters

- **context** (`ContextInterface`) – The context in which to store configuration data that the automagic might populate
- **config_path** (`str`) – Configuration path where the configurable's data under the context's config lives
- **configurable** – The top level configurable whose requirements may need satisfying
- **progress_callback** – An optional function accepting a percentage and optional description to indicate progress during long calculations

Note: The *context* provided here may be different to that provided during initialization. The *context* provided at initialization should be used for local configuration of the automagic itself, the *context* provided during the call is to be populated by the automagic.

Basic initializer that allows configurables to access their own config settings.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

exclusion_list = []

A list of plugin categories (typically operating systems) which the plugin will not operate on

find_requirements(context, config_path, requirement_root, requirement_type, shortcut=True)

Determines if there is actually an unfulfilled *Requirement* waiting.

This ensures we do not carry out an expensive search when there is no need for a particular *Requirement*

Parameters

- **context** (*ContextInterface*) – Context on which to operate
- **config_path** (*str*) – Configuration path of the top-level requirement
- **requirement_root** (*RequirementInterface*) – Top-level requirement whose subrequirements will all be searched
- **requirement_type** (*Union[Tuple[Type[RequirementInterface], ...], Type[RequirementInterface]*) – Type of requirement to find
- **shortcut** (*bool*) – Only returns requirements that live under unsatisfied requirements

Return type

List[Tuple[str, RequirementInterface]]

Returns

A list of tuples containing the config_path, sub_config_path and requirement identifying the unsatisfied *Requirements*

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

List[RequirementInterface]

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

priority = 10

An ordering to indicate how soon this automagic should be run

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

class `StackerLayerInterface`

Bases: *object*

Class that takes a lower layer and attempts to build on it.

`stack_order` determines the order (from low to high) that stacking layers should be attempted lower levels should have lower *stack_orders*

exclusion_list: *List[str]* = []

The list operating systems/first-level plugin hierarchy that should exclude this stacker

classmethod `stack(context, layer_name, progress_callback=None)`

Method to determine whether this builder can operate on the named layer. If so, modify the context appropriately.

Returns the name of any new layer stacked on top of this layer or None. The stacking is therefore strictly linear rather than tree driven.

Configuration options provided by the context are ignored, and defaults are to be used by this method to build a space where possible.

Parameters

- **context** (*ContextInterface*) – Context in which to construct the higher layer
- **layer_name** (*str*) – Name of the layer to stack on top of

- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callback function to indicate progress through a scan (if one is necessary)

Return type`Optional[DataLayerInterface]`**stack_order** = 0

The order in which to attempt stacking, the lower the earlier

classmethod `stacker_slow_warning()`**volatility3.framework.interfaces.configuration module**

The configuration module contains classes and functions for interacting with the configuration and requirement trees.

Volatility plugins can specify a list of requirements (which may have subrequirements, thus forming a requirement tree). These requirement trees can contain values, which are contained in a complementary configuration tree. These two trees act as a protocol between the plugins and users. The plugins provide requirements that must be fulfilled, and the users provide configurations values that fulfill those requirements. Where the user does not provide sufficient configuration values, automatic modules may extend the configuration tree themselves.

CONFIG_SEPARATOR = '.'

Use to specify the separator between configuration hierarchies

class `ClassRequirement(*args, **kwargs)`Bases: `RequirementInterface`

Requires a specific class.

This is used as means to serialize specific classes for `TranslationLayerRequirement` and `SymbolTableRequirement` classes.

Parameters

- **name** – The name of the requirement
- **description** – A short textual description of the requirement
- **default** – The default value for the requirement if no value is provided
- **optional** – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Adds a child to the list of requirements.

Parameters**requirement** (`RequirementInterface`) – The requirement to add as a child-requirement**Return type**`None`**property** `cls: Type | None`

Contains the actual chosen class based on the configuration value's class name.

config_value(*context*, *config_path*, *default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (`ContextInterface`) – the configuration store to find the value for this requirement

- **config_path** (*str*) – the configuration path of the instance of the requirement to be recovered
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None*) – a default value to provide if the requirement’s configuration value is not found

Return type*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]***property default:** *int | bool | bytes | str | List[int | bool | bytes | str] | None*

Returns the default value if one is set.

property description: *str*

A short description of what the Requirement is designed to affect or achieve.

property name: *str*

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

property optional: *bool*

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Removes a child from the list of requirements.

Parameters**requirement** (*RequirementInterface*) – The requirement to remove as a child-requirement**Return type***None***property requirements:** *Dict[str, RequirementInterface]*

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context, config_path*)

Checks to see if a class can be recovered.

Return type*Dict[str, RequirementInterface]***unsatisfied_children**(*context, config_path*)

Method that will validate all child requirements.

Parameters

- **context** (*ContextInterface*) – the context containing the configuration data for this requirement
- **config_path** (*str*) – the configuration path of this instance of the requirement

Return type*Dict[str, RequirementInterface]***Returns**

A dictionary of full configuration paths for each unsatisfied child-requirement

class ConfigurableInterface(*context, config_path*)

Bases: `object`

Class to allow objects to have requirements and read configuration data from the context config tree.

Basic initializer that allows configurables to access their own config settings.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

`List[RequirementInterface]`

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

class ConfigurableRequirementInterface(*name, description=None, default=None, optional=False*)

Bases: [RequirementInterface](#)

Simple Abstract class to provide build_required_config.

Parameters

- **name** ([str](#)) – The name of the requirement
- **description** ([str](#)) – A short textual description of the requirement
- **default** ([Union\[int, bool, bytes, str, List\[Union\[int, bool, bytes, str\]\], None\]](#)) – The default value for the requirement if no value is provided
- **optional** ([bool](#)) – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Adds a child to the list of requirements.

Parameters

requirement ([RequirementInterface](#)) – The requirement to add as a child-requirement

Return type

[None](#)

build_configuration(*context, config_path, value*)

Proxies to a ConfigurableInterface if necessary.

Return type

[HierarchicalDict](#)

config_value(*context, config_path, default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** ([ContextInterface](#)) – the configuration store to find the value for this requirement
- **config_path** ([str](#)) – the configuration path of the instance of the requirement to be recovered
- **default** ([Union\[int, bool, bytes, str, List\[Union\[int, bool, bytes, str\]\], None\]](#)) – a default value to provide if the requirement's configuration value is not found

Return type

[Union\[int, bool, bytes, str, List\[Union\[int, bool, bytes, str\]\], None\]](#)

property default: [int](#) | [bool](#) | [bytes](#) | [str](#) | [List\[int | bool | bytes | str\]](#) | [None](#)

Returns the default value if one is set.

property description: [str](#)

A short description of what the Requirement is designed to affect or achieve.

property name: [str](#)

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

property optional: [bool](#)

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Removes a child from the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to remove as a child-requirement

Return type

None

property requirements: *Dict[str, RequirementInterface]*

Returns a dictionary of all the child requirements, indexed by name.

abstract unsatisfied(*context, config_path*)

Method to validate the value stored at *config_path* for the configuration object against a context.

Returns a list containing its own name (or multiple unsatisfied requirement names) when invalid

Parameters

- **context** (*ContextInterface*) – The context object containing the configuration for this requirement
- **config_path** (*str*) – The configuration path for this requirement to test satisfaction

Return type

Dict[str, RequirementInterface]

Returns

A dictionary of configuration-paths to requirements that could not be satisfied

unsatisfied_children(*context, config_path*)

Method that will validate all child requirements.

Parameters

- **context** (*ContextInterface*) – the context containing the configuration data for this requirement
- **config_path** (*str*) – the configuration path of this instance of the requirement

Return type

Dict[str, RequirementInterface]

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

class ConstructableRequirementInterface(*args, **kwargs)

Bases: *RequirementInterface*

Defines a Requirement that can be constructed based on their own requirements.

This effectively offers a means for serializing specific python types, to be reconstructed based on simple configuration data. Each constructable records a *class* requirement, which indicates the object that will be constructed. That class may have its own requirements (which is why validation of a ConstructableRequirement must happen after the class configuration value has been provided). These values are then provided to the object's constructor by name as arguments (as well as the standard *context* and *config_path* arguments).

Parameters

- **name** – The name of the requirement
- **description** – A short textual description of the requirement

- **default** – The default value for the requirement if no value is provided
- **optional** – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Adds a child to the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to add as a child-requirement

Return type

None

config_value(*context*, *config_path*, *default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (*ContextInterface*) – the configuration store to find the value for this requirement
- **config_path** (*str*) – the configuration path of the instance of the requirement to be recovered
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None*) – a default value to provide if the requirement's configuration value is not found

Return type

Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]

abstract construct(*context*, *config_path*)

Method for constructing within the context any required elements from subrequirements.

Parameters

- **context** (*ContextInterface*) – The context object containing the configuration data for the constructable
- **config_path** (*str*) – The configuration path for the specific instance of this constructable

Return type

None

property default: *int | bool | bytes | str | List[int | bool | bytes | str] | None*

Returns the default value if one is set.

property description: *str*

A short description of what the Requirement is designed to affect or achieve.

property name: *str*

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

property optional: *bool*

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Removes a child from the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to remove as a child-requirement

Return type

None

property requirements: *Dict[str, RequirementInterface]*

Returns a dictionary of all the child requirements, indexed by name.

abstract unsatisfied(*context, config_path*)

Method to validate the value stored at *config_path* for the configuration object against a context.

Returns a list containing its own name (or multiple unsatisfied requirement names) when invalid

Parameters

- **context** (*ContextInterface*) – The context object containing the configuration for this requirement
- **config_path** (*str*) – The configuration path for this requirement to test satisfaction

Return type

Dict[str, RequirementInterface]

Returns

A dictionary of configuration-paths to requirements that could not be satisfied

unsatisfied_children(*context, config_path*)

Method that will validate all child requirements.

Parameters

- **context** (*ContextInterface*) – the context containing the configuration data for this requirement
- **config_path** (*str*) – the configuration path of this instance of the requirement

Return type

Dict[str, RequirementInterface]

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

class HierarchicalDict(*initial_dict=None, separator='.'*)

Bases: *Mapping*

The core of configuration data, it is a mapping class that stores keys within itself, and also stores lower hierarchies.

Parameters

- **initial_dict** (*Dict[str, SimpleTypeRequirement]*) – A dictionary to populate the HierarchicalDict with initially
- **separator** (*str*) – A custom hierarchy separator (defaults to CONFIG_SEPARATOR)

branch(*key*)

Returns the HierarchicalDict housed under the key.

This differs from the data property, in that it is directed by the *key*, and all layers under that key are returned, not just those in that level.

Higher layers are not prefixed with the location of earlier layers, so branching a hierarchy containing *a.b.c.d* on *a.b* would return a hierarchy containing *c.d*, not *a.b.c.d*.

Parameters

key (*str*) – The location within the hierarchy to return higher layers.

Return type

HierarchicalDict

Returns

The HierarchicalDict underneath the specified key (not just the data at that key location in the tree)

clone()

Duplicates the configuration, allowing changes without affecting the original.

Return type

HierarchicalDict

Returns

A duplicate HierarchicalDict of this object

property data: Dict

Returns just the data-containing mappings on this level of the Hierarchy.

generator()

A generator for the data in this level and lower levels of this mapping.

Return type

Generator[*str*, *None*, *None*]

Returns

Returns each item in the top level data, and then all subkeys in a depth first order

get(*k* [, *d*]) → *D*[*k*] if *k* in *D*, else *d*. *d* defaults to *None*.

items() → a set-like object providing a view on *D*'s items

keys() → a set-like object providing a view on *D*'s keys

merge(key, value, overwrite=False)

Acts similarly to splice, but maintains previous values.

If *overwrite* is true, then entries in the new *value* are used over those that exist within *key* already

Parameters

- **key** (*str*) – The location within the hierarchy at which to merge the *value*
- **value** (*HierarchicalDict*) – HierarchicalDict to be merged under the *key* node
- **overwrite** (*bool*) – A boolean defining whether the *value* will be overwritten if it already exists

Return type

None

property separator: str

Specifies the hierarchy separator in use in this HierarchyDict.

splice(key, value)

Splices an existing HierarchicalDictionary under a specific *key*.

This can be thought of as an inverse of *branch()*, although *branch* does not remove the requested hierarchy, it simply returns it.

Return type`None``values()` → an object providing a view on D's values**class RequirementInterface**(*name*, *description=None*, *default=None*, *optional=False*)Bases: `object`

Class that defines a requirement.

A requirement is a means for plugins and other framework components to request specific configuration data. Requirements can either be simple types (such as `SimpleTypeRequirement`, `IntRequirement`, `BytesRequirement` and `StringRequirement`) or complex types (such as `TranslationLayerRequirement`, `SymbolTableRequirement` and `ClassRequirement`)

Parameters

- **name** (`str`) – The name of the requirement
- **description** (`str`) – A short textual description of the requirement
- **default** (`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`) – The default value for the requirement if no value is provided
- **optional** (`bool`) – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Adds a child to the list of requirements.

Parameters**requirement** (`RequirementInterface`) – The requirement to add as a child-requirement**Return type**`None`**config_value**(*context*, *config_path*, *default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (`ContextInterface`) – the configuration store to find the value for this requirement
- **config_path** (`str`) – the configuration path of the instance of the requirement to be recovered
- **default** (`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`) – a default value to provide if the requirement's configuration value is not found

Return type`Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`**property default:** `int | bool | bytes | str | List[int | bool | bytes | str] | None`

Returns the default value if one is set.

property description: `str`

A short description of what the Requirement is designed to affect or achieve.

property name: `str`

The name of the Requirement.

Names cannot contain CONFIG_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

property optional: `bool`

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Removes a child from the list of requirements.

Parameters

requirement (*RequirementInterface*) – The requirement to remove as a child-requirement

Return type

`None`

property requirements: `Dict[str, RequirementInterface]`

Returns a dictionary of all the child requirements, indexed by name.

abstract unsatisfied(*context, config_path*)

Method to validate the value stored at *config_path* for the configuration object against a context.

Returns a list containing its own name (or multiple unsatisfied requirement names) when invalid

Parameters

- **context** (*ContextInterface*) – The context object containing the configuration for this requirement
- **config_path** (*str*) – The configuration path for this requirement to test satisfaction

Return type

`Dict[str, RequirementInterface]`

Returns

A dictionary of configuration-paths to requirements that could not be satisfied

unsatisfied_children(*context, config_path*)

Method that will validate all child requirements.

Parameters

- **context** (*ContextInterface*) – the context containing the configuration data for this requirement
- **config_path** (*str*) – the configuration path of this instance of the requirement

Return type

`Dict[str, RequirementInterface]`

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

class SimpleTypeRequirement(*name, description=None, default=None, optional=False*)

Bases: *RequirementInterface*

Class to represent a single simple type (such as a boolean, a string, an integer or a series of bytes)

Parameters

- **name** (*str*) – The name of the requirement
- **description** (*str*) – A short textual description of the requirement
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]*) – The default value for the requirement if no value is provided
- **optional** (*bool*) – Whether the requirement must be satisfied or not

add_requirement(*requirement*)

Always raises a `TypeError` as instance requirements cannot have children.

config_value(*context*, *config_path*, *default=None*)

Returns the value for this Requirement from its config path.

Parameters

- **context** (*ContextInterface*) – the configuration store to find the value for this requirement
- **config_path** (*str*) – the configuration path of the instance of the requirement to be recovered
- **default** (*Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None*) – a default value to provide if the requirement’s configuration value is not found

Return type

Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]

property default: *int | bool | bytes | str | List[int | bool | bytes | str] | None*

Returns the default value if one is set.

property description: *str*

A short description of what the Requirement is designed to affect or achieve.

instance_type

alias of *bool*

property name: *str*

The name of the Requirement.

Names cannot contain `CONFIG_SEPARATOR` (‘.’ by default) since this is used within the configuration hierarchy.

property optional: *bool*

Whether the Requirement is optional or not.

remove_requirement(*requirement*)

Always raises a `TypeError` as instance requirements cannot have children.

property requirements: *Dict[str, RequirementInterface]*

Returns a dictionary of all the child requirements, indexed by name.

unsatisfied(*context*, *config_path*)

Validates the instance requirement based upon its *instance_type*.

Return type

Dict[str, RequirementInterface]

unsatisfied_children(*context*, *config_path*)

Method that will validate all child requirements.

Parameters

- **context** (*ContextInterface*) – the context containing the configuration data for this requirement
- **config_path** (*str*) – the configuration path of this instance of the requirement

Return type

Dict[str, RequirementInterface]

Returns

A dictionary of full configuration paths for each unsatisfied child-requirement

class VersionableInterface(*args, **kwargs)

Bases: `object`

A class that allows version checking so that plugins can request specific versions of components they made need

This currently includes other Plugins and scanners, but may be extended in the future

All version number should use semantic versioning

version = (0, 0, 0)

parent_path(value)

Returns the parent configuration path from a configuration path.

Return type

`str`

path_depth(path, depth=1)

Returns the *path* up to a certain depth.

Note that *depth* can be negative (such as -x) and will return all elements except for the last x components

Return type

`str`

path_head(value)

Return the top of the configuration path

Return type

`str`

path_join(*args)

Joins configuration paths together.

Return type

`str`

volatility3.framework.interfaces.context module

Defines an interface for contexts, which hold the core components that a plugin will operate upon when running.

These include a *memory* container which holds a series of forest of layers, and a *symbol_space* which contains tables of symbols that can be used to interpret data in a layer. The context also provides some convenience functions, most notably the object constructor function, *object*, which will construct a symbol on a layer at a particular offset.

class ContextInterface

Bases: `object`

All context-like objects must adhere to the following interface.

This interface is present to avoid import dependency cycles.

Initializes the context with a *symbol_space*.

add_layer(layer)

Adds a named translation layer to the context memory.

Parameters

layer (*DataLayerInterface*) – Layer object to be added to the context memory

add_module(*module*)

Adds a named module to the context.

Parameters

module (*ModuleInterface*) – The module to be added to the module object collection

Raises

volatility3.framework.exceptions.VolatilityException – if the module is already present, or has unmet dependencies

clone()

Produce a clone of the context (and configuration), allowing modifications to be made without affecting any mutable objects in the original.

Memory constraints may become an issue for this function depending on how much is actually stored in the context

Return type

ContextInterface

abstract property config: *HierarchicalDict*

Returns the configuration object for this context.

abstract property layers: *LayerContainer*

Returns the memory object for the context.

module(*module_name*, *layer_name*, *offset*, *native_layer_name*=None, *size*=None)

Create a module object.

A module object is associated with a symbol table, and acts like a context, but offsets locations by a known value and looks up symbols, by default within the associated symbol table. It can also be sized should that information be available.

Parameters

- **module_name** (*str*) – The name of the module
- **layer_name** (*str*) – The layer the module is associated with (which layer the module lives within)
- **offset** (*int*) – The initial/base offset of the module (used as the offset for relative symbols)
- **native_layer_name** (*Optional[str]*) – The default *native_layer_name* to use when the module constructs objects
- **size** (*Optional[int]*) – The size, in bytes, that the module occupies from offset location within the layer named *layer_name*

Return type

ModuleInterface

Returns

A module object

abstract property modules: *ModuleContainer*

Returns the memory object for the context.

abstract object(*object_type*, *layer_name*, *offset*, *native_layer_name*=None, *arguments*)**

Object factory, takes a context, symbol, offset and optional *layer_name*.

Looks up the *layer_name* in the context, finds the object template based on the symbol, and constructs an object using the object template on the layer at the offset.

Parameters

- **object_type** (`Union[str, Template]`) – Either a string name of the type, or a Template of the type to be constructed
- **layer_name** (`str`) – The name of the layer on which to construct the object
- **offset** (`int`) – The address within the layer at which to construct the object
- **native_layer_name** (`str`) – The layer this object references (should it be a pointer or similar)

Returns

A fully constructed object

abstract property symbol_space: `SymbolSpaceInterface`

Returns the symbol_space for the context.

This object must support the `SymbolSpaceInterface`

class ModuleContainer(*modules=None*)

Bases: `Mapping`

Container for multiple layers of data.

add_module(*module*)

Adds a module to the module collection

This will throw an exception if the required dependencies are not met

Parameters

module (`ModuleInterface`) – the module to add to the list of modules (based on module.name)

Return type

`None`

free_module_name(*prefix='module'*)

Returns an unused table name to ensure no collision occurs when inserting a symbol table.

Return type

`str`

get(*k[, d]*) → *D*[*k*] if *k* in *D*, else *d*. *d* defaults to `None`.

get_modules_by_symbol_tables(*symbol_table*)

Returns the modules which use the specified symbol table name

Return type

`Iterable[str]`

items() → a set-like object providing a view on *D*'s items

keys() → a set-like object providing a view on *D*'s keys

values() → an object providing a view on *D*'s values

class ModuleInterface(*context, config_path, name*)

Bases: `ConfigurableInterface`

Maintains state concerning a particular loaded module in memory.

This object is OS-independent.

Constructs a new os-independent module.

Parameters

- **context** (*ContextInterface*) – The context within which this module will exist
- **config_path** (*str*) – The path within the context’s configuration tree
- **name** (*str*) – The name of the module

build_configuration()

Builds the configuration dictionary for this specific Module

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

Context that the module uses.

get_absolute_symbol_address(*name*)

Returns the absolute address of the symbol within this module

Return type

int

get_enumeration(*name*)

Returns an enumeration from the module’s symbol table.

Return type

Template

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

List[RequirementInterface]

get_symbol(*name*)

Returns a symbol object from the module’s symbol table.

Return type

SymbolInterface

get_symbols_by_absolute_location(*offset, size=0*)

Returns the symbols within table_name (or this module if not specified) that live at the specified absolute offset provided.

Return type

List[str]

get_type(*name*)

Returns a type from the module’s symbol table.

Return type

Template

has_enumeration(*name*)

Determines whether an enumeration is present in the module's symbol table.

Return type

`bool`

has_symbol(*name*)

Determines whether a symbol is present in the module's symbol table.

Return type

`bool`

has_type(*name*)

Determines whether a type is present in the module's symbol table.

Return type

`bool`

property layer_name: `str`

Layer name in which the Module resides.

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property name: `str`

The name of the constructed module.

abstract object(*object_type*, *offset=None*, *native_layer_name=None*, *absolute=False*, ***kwargs*)

Returns an object created using the `symbol_table_name` and `layer_name` of the Module.

Parameters

- **object_type** (`str`) – The name of object type to construct (using the module's `symbol_table`)
- **offset** (`int`) – the offset (unless `absolute` is set) from the start of the module
- **native_layer_name** (*Optional*[`str`]) – The native layer for objects that reference a different layer (if not the default provided during module construction)
- **absolute** (`bool`) – A boolean specifying whether the offset is absolute within the layer, or relative to the start of the module

Return type

ObjectInterface

Returns

The constructed object

abstract object_from_symbol(*symbol_name*, *native_layer_name*=None, *absolute*=False, *object_type*=None, ***kwargs*)

Returns an object created using the *symbol_table_name* and *layer_name* of the Module.

Parameters

- **symbol_name** (*str*) – The name of a symbol (that must be present in the module’s symbol table). The symbol’s associated type will be used to construct an object at the symbol’s offset.
- **native_layer_name** (*Optional[str]*) – The native layer for objects that reference a different layer (if not the default provided during module construction)
- **absolute** (*bool*) – A boolean specifying whether the offset is absolute within the layer, or relative to the start of the module
- **object_type** (*Union[str, ObjectInterface, None]*) – Override for the type from the symbol to use (or if the symbol type is missing)

Return type

ObjectInterface

Returns

The constructed object

property offset: *int*

Returns the offset that the module resides within the layer of *layer_name*.

property symbol_table_name: *str*

The name of the symbol table associated with this module

symbols()

Lists the symbols contained in the symbol table for this module

Return type

List

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

volatility3.framework.interfaces.layers module

Defines layers for containing data.

One layer may combine other layers, map data based on the data itself, or map a procedure (such as decryption) across another layer of data.

class DataLayerInterface(*context, config_path, name, metadata=None*)

Bases: *ConfigurableInterface*

A Layer that directly holds data (and does not translate it).

This is effectively a leaf node in a layer tree. It directly accesses a data source and exposes it within volatility.

Basic initializer that allows configurables to access their own config settings.

property address_mask: *int*

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

property dependencies: *List[str]*

A list of other layer names required by this layer.

Note: DataLayers must never define other layers

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

None

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

List[RequirementInterface]

abstract `is_valid(offset, length=1)`

Returns a boolean based on whether the entire chunk of data (from offset to length) is valid or not.

Parameters

- **offset** (`int`) – The address to start determining whether bytes are readable/valid
- **length** (`int`) – The number of bytes from offset of which to test the validity

Return type

`bool`

Returns

Whether the bytes are valid and accessible

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

abstract property `maximum_address: int`

Returns the maximum valid address of the space.

property metadata: `Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

abstract property `minimum_address: int`

Returns the minimum valid address of the space.

property name: `str`

Returns the layer name.

abstract `read(offset, length, pad=False)`

Reads an offset for length bytes and returns ‘bytes’ (not ‘str’) of length size.

If there is a fault of any kind (such as a page fault), an exception will be thrown unless pad is set, in which case the read errors will be replaced by null characters.

Parameters

- **offset** (`int`) – The offset at which to begin reading within the layer
- **length** (`int`) – The number of bytes to read within the layer
- **pad** (`bool`) – A boolean indicating whether exceptions should be raised or bad bytes replaced with null characters

Return type

`bytes`

Returns

The bytes read from the layer, starting at offset for length bytes

scan(*context*, *scanner*, *progress_callback*=None, *sections*=None)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (*ContextInterface*) – The context containing the data layer
- **scanner** (*ScannerInterface*) – The constructed Scanner object to be applied
- **progress_callback** (Optional[Callable[[float, str], None]]) – Method that is called periodically during scanning to update progress
- **sections** (Iterable[Tuple[int, int]]) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

Iterable[Any]

Returns

The output iterable from the scanner object having been run against the layer

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

abstract write(*offset*, *data*)

Writes a chunk of data at offset.

Any unavailable sections in the underlying bases will cause an exception to be thrown. Note: Writes are not guaranteed atomic, therefore some data may have been written, even if an exception is thrown.

Return type

None

class DummyProgress

Bases: *object*

A class to emulate Multiprocessing/threading Value objects.

class LayerContainer

Bases: *Mapping*

Container for multiple layers of data.

add_layer(*layer*)

Adds a layer to memory model.

This will throw an exception if the required dependencies are not met

Parameters

layer (*DataLayerInterface*) – the layer to add to the list of layers (based on layer.name)

Return type

None

check_cycles()

Runs through the available layers and identifies if there are cycles in the DAG.

Return type

None

del_layer(name)

Removes the layer called name.

This will throw an exception if other layers depend upon this layer

Parameters

name (*str*) – The name of the layer to delete

Return type

None

free_layer_name(prefix='layer')

Returns an unused layer name to ensure no collision occurs when inserting a layer.

Parameters

prefix (*str*) – A descriptive string with which to prefix the layer name

Return type

str

Returns

A string containing a name, prefixed with prefix, not currently in use within the LayerContainer

get(*k*, *d*) → *D*[*k*] if *k* in *D*, else *d*. *d* defaults to *None*.

items() → a set-like object providing a view on *D*'s items

keys() → a set-like object providing a view on *D*'s keys

read(layer, offset, length, pad=False)

Reads from a particular layer at offset for length bytes.

Returns 'bytes' not 'str'

Parameters

- **layer** (*str*) – The name of the layer to read from
- **offset** (*int*) – Where to begin reading within the layer
- **length** (*int*) – How many bytes to read from the layer
- **pad** (*bool*) – Whether to raise exceptions or return null bytes when errors occur

Return type

bytes

Returns

The result of reading from the requested layer

values() → an object providing a view on D's values

write(layer, offset, data)

Writes to a particular layer at offset for length bytes.

Return type

`None`

class ScannerInterface

Bases: `VersionableInterface`

Class for layer scanners that return locations of particular values from within the data.

These are designed to be given a chunk of data and return a generator which yields any found items. They should NOT perform complex/time-consuming tasks, these should be carried out by the consumer of the generator on the items returned.

They will be provided all *available* data (therefore not necessarily contiguous) in ascending offset order, in chunks no larger than `chunk_size + overlap` where `overlap` is the amount of data read twice once at the end of an earlier chunk and once at the start of the next chunk.

It should be noted that the scanner can maintain state if necessary. Scanners should balance the size of chunk based on the amount of time scanning the chunk will take (ie, do not set an excessively large chunksize and try not to take a significant amount of time in the `__call__` method).

Scanners must NOT return results found *after* `self.chunk_size` (ie, entirely contained within the overlap). It is the responsibility of the scanner not to return such duplicate results.

Scanners can mark themselves as `thread_safe`, if they do not require state in either their own class or the context. This will allow the scanner to be run in parallel against multiple blocks.

property context: `ContextInterface` | `None`

property layer_name: `str` | `None`

thread_safe = `False`

version = `(0, 0, 0)`

class TranslationLayerInterface(context, config_path, name, metadata=None)

Bases: `DataLayerInterface`

Provides a layer that translates or transforms another layer or layers.

Translation layers always depend on another layer (typically translating offsets in a virtual offset space into a smaller physical offset space).

Basic initializer that allows configurables to access their own config settings.

property address_mask: `int`

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

abstract property dependencies: *List[str]*

Returns a list of layer names that this layer translates onto.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

None

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

List[RequirementInterface]

abstract is_valid(offset, length=1)

Returns a boolean based on whether the entire chunk of data (from offset to length) is valid or not.

Parameters

- **offset** (*int*) – The address to start determining whether bytes are readable/valid
- **length** (*int*) – The number of bytes from offset of which to test the validity

Return type

bool

Returns

Whether the bytes are valid and accessible

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

abstract mapping(*offset, length, ignore_errors=False*)

Returns a sorted iterable of (offset, sublength, mapped_offset, mapped_length, layer) mappings.

ignore_errors will provide all available maps with gaps, but their total length may not add up to the requested length. This allows translation layers to provide maps of contiguous regions in one layer.

Return type

`Iterable[Tuple[int, int, int, int, str]]`

abstract property maximum_address: `int`

Returns the maximum valid address of the space.

property metadata: `Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

abstract property minimum_address: `int`

Returns the minimum valid address of the space.

property name: `str`

Returns the layer name.

read(*offset, length, pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

`bytes`

scan(*context, scanner, progress_callback=None, sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (`ContextInterface`) – The context containing the data layer
- **scanner** (`ScannerInterface`) – The constructed Scanner object to be applied
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – Method that is called periodically during scanning to update progress
- **sections** (`Iterable[Tuple[int, int]]`) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

`Iterable[Any]`

Returns

The output iterable from the scanner object having been run against the layer

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

write(*offset*, *value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

`None`

volatility3.framework.interfaces.objects module

Objects are the core of volatility, and provide pythonic access to interpreted values of data from a layer.

class `ObjectInformation`(*layer_name*, *offset*, *member_name=None*, *parent=None*, *native_layer_name=None*, *size=None*)

Bases: `ReadOnlyMapping`

Contains common information useful/pertinent only to an individual object (like an instance)

This typically contains information such as the layer the object belongs to, the offset where it was constructed, and if it is a subordinate object, its parent.

This is primarily used to reduce the number of parameters passed to object constructors and keep them all together in a single place. These values are based on the `ReadOnlyMapping` class, to prevent their modification.

Constructs a container for basic information about an object.

Parameters

- **layer_name** (`str`) – Layer from which the data for the object will be read
- **offset** (`int`) – Offset within the layer at which the data for the object will be read
- **member_name** (`Optional[str]`) – If the object was accessed as a member of a parent object, this was the name used to access it
- **parent** (`Optional[ObjectInterface]`) – If the object was accessed as a member of a parent object, this is the parent object
- **native_layer_name** (`Optional[str]`) – If this object references other objects (such as a pointer), what layer those objects live in
- **size** (`Optional[int]`) – The size that the whole structure consumes in bytes

get(*k*, *d*) → *D*[*k*] if *k* in *D*, else *d*. *d* defaults to `None`.

items() → a set-like object providing a view on *D*'s items

keys() → a set-like object providing a view on *D*'s keys

values() → an object providing a view on *D*'s values

class `ObjectInterface`(*context*, *type_name*, *object_info*, ***kwargs*)

Bases: `object`

A base object required to be the ancestor of every object used in volatility.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object

- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *object*

A container for proxied methods that the ObjectTemplate of this object will call. This is primarily to keep methods together for easy organization/management, there is no significant need for it to be a separate class.

The methods of this class *must* be class methods rather than standard methods, to allow for code reuse. Each method also takes a template since the templates may contain the necessary data about the yet-to-be-constructed object. It allows objects to control how their templates respond without needing to write new templates for each and every potential object type.

abstract classmethod child_template(*template, child*)

Returns the template of the child member from the parent.

Return type

Template

abstract classmethod children(*template*)

Returns the children of the template.

Return type

List[Template]

abstract classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

abstract classmethod relative_child_offset(*template, child*)

Returns the relative offset from the head of the parent data to the child member.

Return type

int

abstract classmethod replace_child(*template, old_child, new_child*)

Substitutes the old_child for the new_child.

Return type

None

abstract classmethod size(*template*)

Returns the size of the template object.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table

- **KeyError** – If the `table_name` is not valid within the object's context

Return type`str`**has_member**(*member_name*)Returns whether the object would contain a member called `member_name`.**Parameters****member_name** (`str`) – Name to test whether a member exists within the type structure**Return type**`bool`**has_valid_member**(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters**member_name** (`str`) – Name of the member to test access to determine if the member is valid or not**Return type**`bool`**has_valid_members**(*member_names*)Returns whether the object has all of the members listed in `member_names`**Parameters****member_names** (`List[str]`) – List of names to test as to members with those names validity**Return type**`bool`**property vol:** `ReadOnlyMapping`

Returns the volatility specific object information.

abstract write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class `ReadOnlyMapping`(*dictionary*)Bases: `Mapping`

A read-only mapping of various values that offer attribute access as well.

This ensures that the data stored in the mapping should not be modified, making an immutable mapping.

get(*k*, *d*) → `D[k]` if *k* in *D*, else *d*. *d* defaults to `None`.**items**() → a set-like object providing a view on *D*'s items**keys**() → a set-like object providing a view on *D*'s keys**values**() → an object providing a view on *D*'s values**class** `Template`(*type_name*, ***arguments*)Bases: `object`

Class for all Factories that take offsets, and data layers and produce objects.

This is effectively a class for currying object calls. It creates a callable that can be called with the following parameters:

Parameters

- **context** – The context containing the memory layers and symbols required to construct the object
- **object_info** – Basic information about the object, see the ObjectInformation class for more information

Returns

The constructed object

The keyword arguments handed to the constructor, along with the `type_name` are stored for later retrieval. These will be access as `object.vol.<keyword>` or `template.vol.<keyword>` for each object and should contain as least the basic information that each object will require before it is instantiated (so *offset* and *parent* are explicitly not recorded here). This dictionary can be updated after construction, but any changes made after that point will *not* be cloned. This is so that templates such as those for string objects may contain different length limits, without affecting all other strings using the same template from a SymbolTable, constructed at resolution time and then cached.

Stores the keyword arguments for later object creation.

abstract child_template(*child*)

Returns the *child* member template from its parent.

Return type

Template

property children: `List[Template]`

The children of this template (such as member types, sub-types and base-types where they are relevant).

Used to traverse the template tree.

clone()

Returns a copy of the original Template as constructed (without *update_vol* additions having been made)

Return type

Template

abstract has_member(*member_name*)

Returns whether the object would contain a member called *member_name*

Return type

`bool`

abstract relative_child_offset(*child*)

Returns the relative offset of the *child* member from its parent offset.

Return type

`int`

abstract replace_child(*old_child*, *new_child*)

Replaces *old_child* with *new_child* in the list of children.

Return type

`None`

abstract property size: `int`

Returns the size of the template.

update_vol(***new_arguments*)

Updates the keyword arguments with values that will **not** be carried across to clones.

Return type

`None`

property vol: [*ReadOnlyMapping*](#)

Returns a volatility information object, much like the [*ObjectInformation*](#) provides.

volatility3.framework.interfaces.plugins module

Plugins are the *functions* of the volatility framework.

They are called and carry out some algorithms on data stored in layers using objects constructed from symbols.

class FileHandlerInterface(*filename*)

Bases: [*RawIOBase*](#)

Class for storing Files in the plugin as a means to output a file when necessary.

This can be used as ContextManager that will close/produce the file automatically when exiting the context block

Creates a FileHandler

Parameters

filename (*str*) – The requested name of the filename for the data

abstract close()

Method that commits the file and fixes the final filename for use

closed

fileno()

Returns underlying file descriptor if one exists.

OSError is raised if the IO object does not use a file descriptor.

flush()

Flush write buffers, if applicable.

This is not implemented for read-only and non-blocking streams.

isatty()

Return whether this is an ‘interactive’ stream.

Return False if it can’t be determined.

property preferred_filename

The preferred filename to save the data to. Until this file has been written, this value may not be the final filename the data is written to.

read(*size=-1, /*)

readable()

Return whether object was opened for reading.

If False, read() will raise OSError.

readall()

Read until EOF, using multiple read() call.

readinto()

readline(*size=-1, /*)

Read and return a line from the stream.

If size is specified, at most size bytes will be read.

The line terminator is always b'n' for binary files; for text files, the newlines argument to open can be used to select the line terminator(s) recognized.

readlines(*hint=-1, /*)

Return a list of lines from the stream.

hint can be specified to control the number of lines read: no more lines will be read if the total size (in bytes/characters) of all lines so far exceeds hint.

static sanitize_filename(*filename*)

Sanitizes the filename to ensure only a specific whitelist of characters is allowed through

Return type

`str`

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

Change the stream position to the given byte offset.

offset

The stream position, relative to 'whence'.

whence

The relative position to seek from.

The offset is interpreted relative to the position indicated by whence. Values for whence are:

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

Return the new absolute position.

seekable()

Return whether object supports random access.

If False, `seek()`, `tell()` and `truncate()` will raise `OSError`. This method may need to do a test `seek()`.

tell()

Return current stream position.

truncate()

Truncate file to size bytes.

File pointer is left unchanged. Size defaults to the current IO position as reported by `tell()`. Returns the new size.

writable()

Return whether object was opened for writing.

If False, `write()` will raise `OSError`.

write()

writelines(*lines*, /)

Write a list of lines to stream.

Line separators are not added, so it is usual for each of the lines provided to have a line separator at the end.

class PluginInterface(*context*, *config_path*, *progress_callback*=None)

Bases: [ConfigurableInterface](#), [VersionableInterface](#)

Class that defines the basic interface that all Plugins must maintain.

The constructor must only take a *context* and *config_path*, so that plugins can be launched automatically. As such all configuration information must be provided through the requirements and configuration information in the context it is passed.

Parameters

- **context** ([ContextInterface](#)) – The context that the plugin will operate within
- **config_path** ([str](#)) – The path to configuration data within the context configuration data
- **progress_callback** ([Optional](#)[[Callable](#)[[[float](#), [str](#)], None]]) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

[HierarchicalDict](#)

property config: [HierarchicalDict](#)

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: [str](#)

The configuration path on which this configurable lives.

property context: [ContextInterface](#)

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

[List](#)[[RequirementInterface](#)]

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** ([ContextInterface](#)) – The context in which to store the new configuration
- **base_config_path** ([str](#)) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property open**

Returns a context manager and thus can be called like open

abstract run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Return type`TreeGrid`**Returns**

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type`None`**classmethod unsatisfied(context, config_path)**

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.framework.interfaces.renderers module

All plugins output a TreeGrid object which must then be rendered (either by a GUI, or as text output, html output or in some other form).

This module defines both the output format (*TreeGrid*) and the renderer interface which can interact with a TreeGrid to produce suitable output.

class BaseAbsentValue

Bases: `object`

Class that represents values which are not present for some reason.

class Column(name, type)

Bases: `tuple`

Create new instance of Column(name, type)

count(*value*, /)

Return number of occurrences of value.

index(*value*, *start*=0, *stop*=9223372036854775807, /)

Return first index of value.

Raises ValueError if the value is not present.

name: `str`

Alias for field number 0

type: `Any`

Alias for field number 1

class `ColumnSortKey`

Bases: `object`

ascending: `bool = True`

class `Disassembly`(*data*, *offset*=0, *architecture*='intel64')

Bases: `object`

A class to indicate that the bytes provided should be disassembled (based on the architecture)

possible_architectures = ['intel', 'intel64', 'arm', 'arm64']

class `Renderer`(*options*=None)

Bases: `object`

Class that defines the interface that all output renderers must support.

Accepts an options object to configure the renderers.

abstract `get_render_options()`

Returns a list of rendering options.

Return type

`List[Any]`

abstract `render`(*grid*)

Takes a grid object and renders it based on the object's preferences.

Return type

`None`

class `TreeGrid`(*columns*, *generator*)

Bases: `object`

Class providing the interface for a TreeGrid (which contains TreeNodes)

The structure of a TreeGrid is designed to maintain the structure of the tree in a single object. For this reason each TreeNode does not hold its children, they are managed by the top level object. This leaves the Nodes as simple data carries and prevents them being used to manipulate the tree as a whole. This is a data structure, and is not expected to be modified much once created.

Carrying the children under the parent makes recursion easier, but then every node is its own little tree and must have all the supporting tree functions. It also allows for a node to be present in several different trees, and to create cycles.

Constructs a TreeGrid object using a specific set of columns.

The TreeGrid itself is a root element, that can have children but no values. The TreeGrid does *not* contain any information about formatting, these are up to the renderers and plugins.

Parameters

- **columns** (`List[Tuple[str, Union[Type[int], Type[str], Type[float], Type[bytes], Type[datetime], Type[BaseAbsentValue], Type[Disassembly]]]]`) – A list of column tuples made up of (name, type).
- **generator** (`Generator`) – An iterable containing row for a tree grid, each row contains a indent level followed by the values for each column in order.

```
base_types: ClassVar[Tuple] = (<class 'int'>, <class 'str'>, <class 'float'>,
<class 'bytes'>, <class 'datetime.datetime'>, <class
'volatility3.framework.interfaces.renderers.Disassembly'>)
```

abstract children(*node*)

Returns the subnodes of a particular node in order.

Return type

`List[TreeNode]`

abstract property columns: List[Column]

Returns the available columns and their ordering and types.

abstract is_ancestor(*node*, *descendant*)

Returns true if descendant is a child, grandchild, etc of node.

Return type

`bool`

abstract max_depth()

Returns the maximum depth of the tree.

Return type

`int`

static path_depth(*node*)

Returns the path depth of a particular node.

Return type

`int`

abstract populate(*function*=None, *initial_accumulator*=None, *fail_on_errors*=True)

Populates the tree by consuming the TreeGrid's construction generator Func is called on every node, so can be used to create output on demand.

This is equivalent to a one-time visit.

Return type

`Optional[Exception]`

abstract property populated: bool

Indicates that population has completed and the tree may now be manipulated separately.

abstract static sanitize_name(*text*)

Method used to sanitize column names for TreeNodes.

Return type

`str`

abstract values(*node*)

Returns the values for a particular node.

The values returned are mutable,

Return type

`Tuple[Union[Type[int], Type[str], Type[float], Type[bytes], Type[datetime], Type[BaseAbsentValue], Type[Disassembly]], ...]`

abstract visit(*node*, *function*, *initial_accumulator*, *sort_key*=None)

Visits all the nodes in a tree, calling function on each one.

function should have the signature function(node, accumulator) and return new_accumulator. If accumulators are not needed, the function must still accept a second parameter.

The order of that the nodes are visited is always depth first, however, the order children are traversed can be set based on a sort_key function which should accept a node's values and return something that can be sorted to receive the desired order (similar to the sort/sorted key).

If node is None, then the root node is used.

Parameters

- **node** (`Optional[TreeNode]`) – The initial node to be visited
- **function** (`Callable[[TreeNode, TypeVar(_Type)], TypeVar(_Type)]`) – The visitor to apply to the nodes under the initial node
- **initial_accumulator** (`TypeVar(_Type)`) – An accumulator that allows data to be transferred between one visitor call to the next
- **sort_key** (`ColumnSortKey`) – Information about the sort order of columns in order to determine the ordering of results

Return type

`None`

class `TreeNode`(*path*, *treegrid*, *parent*, *values*)

Bases: `Sequence`

Initializes the `TreeNode`.

count(*value*) → integer -- return number of occurrences of value

index(*value*[, *start*[, *stop*]]) → integer -- return first index of value.

Raises `ValueError` if the value is not present.

Supporting start and stop arguments is optional, but recommended.

abstract property parent: `TreeNode` | `None`

Returns the parent node of this node or None.

abstract property path: `str`

Returns a path identifying string.

This should be seen as opaque by external classes, Parsing of path locations based on this string are not guaranteed to remain stable.

abstract path_changed(*path*, *added*=False)

Updates the path based on the addition or removal of a node higher up in the tree.

This should only be called by the containing `TreeGrid` and expects to only be called for affected nodes.

Return type

`None`

abstract property path_depth: `int`

Return the path depth of the current node.

abstract property values: `List[Type[int] | Type[str] | Type[float] | Type[bytes] | Type[datetime] | Type[BaseAbsentValue] | Type[Disassembly]]`

Returns the list of values from the particular node, based on column index.

volatility3.framework.interfaces.symbols module

Symbols provide structural information about a set of bytes.

class BaseSymbolTableInterface(*name*, *native_types*, *table_mapping*=None, *class_types*=None)

Bases: `object`

The base interface, inherited by both NativeTables and SymbolTables.

native_types is a NativeTableInterface used for native types for the particular loaded symbol table *table_mapping* allows tables referenced by symbols to be remapped to a different table name if necessary

Note: *table_mapping* is a rarely used feature (since symbol tables are typically self-contained)

Parameters

- **name** (`str`) – Name of the symbol table
- **native_types** (`NativeTableInterface`) – The native symbol table used to resolve any base/native types
- **table_mapping** (`Optional[Dict[str, str]]`) – A dictionary mapping names of tables (which when present within the table will be changed to the mapped table)
- **class_types** (`Optional[Mapping[str, Type[ObjectInterface]]]`) – A dictionary of types and classes that should be instantiated instead of Struct to construct them

clear_symbol_cache()

Clears the symbol cache of this symbol table.

Return type

`None`

del_type_class(*name*)

Removes the associated class override for a specific Symbol type.

Return type

`None`

property enumerations: `Iterable[Any]`

Returns an iterator of the Enumeration names.

get_symbol(*name*)

Resolves a symbol name into a symbol object.

If the symbol isn't found, it raises a SymbolError exception

Return type

`SymbolInterface`

get_symbol_type(*name*)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type`Optional[Template]`**get_symbols_by_location**(*offset*, *size=0*)

Returns the name of all symbols in this table that live at a particular offset.

Return type`Iterable[str]`**get_symbols_by_type**(*type_name*)

Returns the name of all symbols in this table that have type matching *type_name*.

Return type`Iterable[str]`**get_type**(*name*)

Resolves a symbol name into an object template.

If the symbol isn't found it raises a `SymbolError` exception

Return type`Template`**get_type_class**(*name*)

Returns the class associated with a Symbol type.

Return type`Type[ObjectInterface]`**property natives:** `NativeTableInterface`

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name*, *clazz*)

Calls the `set_type_class` function but does not throw an exception. Returns whether setting the type class was successful. :type name: `str` :param name: The name of the type to override the class for :type clazz: `Type[ObjectInterface]` :param clazz: The actual class to override for the provided type name

Return type`bool`**set_type_class**(*name*, *clazz*)

Overrides the object class for a specific Symbol type.

Name *must* be present in `self.types`

Parameters

- **name** (`str`) – The name of the type to override the class for
- **clazz** (`Type[ObjectInterface]`) – The actual class to override for the provided type name

Return type`None`**property symbols:** `Iterable[str]`

Returns an iterator of the Symbol names.

property types: `Iterable[str]`

Returns an iterator of the Symbol type names.

class MetadataInterface(*json_data*)

Bases: `object`

Interface for accessing metadata stored within a symbol table.

Constructor that accepts *json_data*.

class NativeTableInterface(*name*, *native_types*, *table_mapping=None*, *class_types=None*)

Bases: `BaseSymbolTableInterface`

Class to distinguish NativeSymbolLists from other symbol lists.

Parameters

- **name** (`str`) – Name of the symbol table
- **native_types** (`NativeTableInterface`) – The native symbol table used to resolve any base/native types
- **table_mapping** (`Optional[Dict[str, str]]`) – A dictionary mapping names of tables (which when present within the table will be changed to the mapped table)
- **class_types** (`Optional[Mapping[str, Type[ObjectInterface]]]`) – A dictionary of types and classes that should be instantiated instead of Struct to construct them

clear_symbol_cache()

Clears the symbol cache of this symbol table.

Return type

`None`

del_type_class(*name*)

Removes the associated class override for a specific Symbol type.

Return type

`None`

property enumerations: `Iterable[str]`

Returns an iterator of the Enumeration names.

get_enumeration(*name*)

Return type

`Template`

get_symbol(*name*)

Resolves a symbol name into a symbol object.

If the symbol isn't found, it raises a SymbolError exception

Return type

`SymbolInterface`

get_symbol_type(*name*)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type

`Optional[Template]`

get_symbols_by_location(*offset*, *size=0*)

Returns the name of all symbols in this table that live at a particular offset.

Return type`Iterable[str]`**get_symbols_by_type**(*type_name*)

Returns the name of all symbols in this table that have type matching *type_name*.

Return type`Iterable[str]`**get_type**(*name*)

Resolves a symbol name into an object template.

If the symbol isn't found it raises a `SymbolError` exception

Return type`Template`**get_type_class**(*name*)

Returns the class associated with a Symbol type.

Return type`Type[ObjectInterface]`**property natives:** `NativeTableInterface`

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name*, *clazz*)

Calls the `set_type_class` function but does not throw an exception. Returns whether setting the type class was successful. :type *name*: `str` :param *name*: The name of the type to override the class for :type *clazz*: `Type[ObjectInterface]` :param *clazz*: The actual class to override for the provided type name

Return type`bool`**set_type_class**(*name*, *clazz*)

Overrides the object class for a specific Symbol type.

Name *must* be present in `self.types`

Parameters

- **name** (`str`) – The name of the type to override the class for
- **clazz** (`Type[ObjectInterface]`) – The actual class to override for the provided type name

Return type`None`**property symbols:** `Iterable[str]`

Returns an iterator of the Symbol names.

property types: `Iterable[str]`

Returns an iterator of the Symbol type names.

class SymbolInterface(*name*, *address*, *type*=None, *constant_data*=None)

Bases: `object`

Contains information about a named location in a program's memory.

Parameters

- **name** (`str`) – Name of the symbol

- **address** (`int`) – Numeric address value of the symbol
- **type** (`Optional[Template]`) – Optional type structure information associated with the symbol
- **constant_data** (`Optional[bytes]`) – Potential constant data the symbol points at

property address: `int`

Returns the relative address of the symbol within the compilation unit.

property constant_data: `bytes | None`

Returns any constant data associated with the symbol.

property name: `str`

Returns the name of the symbol.

property type: `Template | None`

Returns the type that the symbol represents.

property type_name: `str | None`

Returns the name of the type that the symbol represents.

class SymbolSpaceInterface

Bases: `Mapping`

An interface for the container that holds all the symbol-containing tables for use within a context.

abstract append(*value*)

Adds a `symbol_list` to the end of the space.

Return type

`None`

abstract clear_symbol_cache(*table_name*)

Clears the symbol cache for the specified table name. If no table name is specified, the caches of all symbol tables are cleared.

Return type

`None`

free_table_name(*prefix='layer'*)

Returns an unused table name to ensure no collision occurs when inserting a symbol table.

Return type

`str`

get(*k*, *d*) → *D*[*k*] if *k* in *D*, else *d*. *d* defaults to `None`.

abstract get_enumeration(*enum_name*)

Look-up an enumeration across all the contained symbol tables.

Return type

`Template`

abstract get_symbol(*symbol_name*)

Look-up a symbol name across all the contained symbol tables.

Return type

`SymbolInterface`

abstract `get_symbols_by_location(offset, size=0, table_name=None)`

Returns all symbols that exist at a specific relative address.

Return type

`Iterable[str]`

abstract `get_symbols_by_type(type_name)`

Returns all symbols based on the type of the symbol.

Return type

`Iterable[str]`

abstract `get_type(type_name)`

Look-up a type name across all the contained symbol tables.

Return type

`Template`

abstract `has_enumeration(name)`

Determines whether an enumeration choice exists in the contained symbol tables.

Return type

`bool`

abstract `has_symbol(name)`

Determines whether a symbol exists in the contained symbol tables.

Return type

`bool`

abstract `has_type(name)`

Determines whether a type exists in the contained symbol tables.

Return type

`bool`

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

values() → an object providing a view on D's values

class `SymbolTableInterface(context, config_path, name, native_types, table_mapping=None, class_types=None)`

Bases: `BaseSymbolTableInterface`, `ConfigurableInterface`, `ABC`

Handles a table of symbols.

Instantiates an `SymbolTable` based on an `IntermediateSymbolFormat` JSON file. This is validated against the appropriate schema.

Parameters

- **context** (`ContextInterface`) – The volatility context for the symbol table
- **config_path** (`str`) – The configuration path for the symbol table
- **name** (`str`) – The name for the symbol table (this is used in symbols e.g. `table!symbol`)
- **isf_url** – The URL pointing to the ISF file location
- **native_types** (`NativeTableInterface`) – The `NativeSymbolTable` that contains the native types for this symbol table

- **table_mapping** (`Optional[Dict[str, str]]`) – A dictionary linking names referenced in the file with symbol tables in the context
- **class_types** (`Optional[Mapping[str, Type[ObjectInterface]]]`) – A dictionary of type names and classes that override StructType when they are instantiated

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

clear_symbol_cache()

Clears the symbol cache of this symbol table.

Return type

None

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

del_type_class(name)

Removes the associated class override for a specific Symbol type.

Return type

None

property enumerations: *Iterable[Any]*

Returns an iterator of the Enumeration names.

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

List[RequirementInterface]

get_symbol(name)

Resolves a symbol name into a symbol object.

If the symbol isn't found, it raises a SymbolError exception

Return type

SymbolInterface

get_symbol_type(name)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type

Optional[Template]

get_symbols_by_location(*offset*, *size=0*)

Returns the name of all symbols in this table that live at a particular offset.

Return type

`Iterable[str]`

get_symbols_by_type(*type_name*)

Returns the name of all symbols in this table that have type matching *type_name*.

Return type

`Iterable[str]`

get_type(*name*)

Resolves a symbol name into an object template.

If the symbol isn't found it raises a `SymbolError` exception

Return type

`Template`

get_type_class(*name*)

Returns the class associated with a Symbol type.

Return type

`Type[ObjectInterface]`

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from *kwargs*.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property natives: `NativeTableInterface`

Returns None or a `NativeTable` for handling space specific native types.

optional_set_type_class(*name*, *clazz*)

Calls the `set_type_class` function but does not throw an exception. Returns whether setting the type class was successful. :type name: `str` :param name: The name of the type to override the class for :type clazz: `Type[ObjectInterface]` :param clazz: The actual class to override for the provided type name

Return type

`bool`

set_type_class(*name*, *clazz*)

Overrides the object class for a specific Symbol type.

Name *must* be present in `self.types`

Parameters

- **name** (`str`) – The name of the type to override the class for
- **clazz** (`Type[ObjectInterface]`) – The actual class to override for the provided type name

Return type`None`**property symbols:** `Iterable[str]`

Returns an iterator of the Symbol names.

property types: `Iterable[str]`

Returns an iterator of the Symbol type names.

classmethod unsatisfied(`context, config_path`)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`**volatility3.framework.layers package****Subpackages****volatility3.framework.layers.codecs package**

Codecs used for encoding or decoding data should live here

volatility3.framework.layers.scanners package**class BytesScanner**(`needle`)Bases: `ScannerInterface`**property context:** `ContextInterface | None`**property layer_name:** `str | None`**thread_safe** = `True`**version** = `(0, 0, 0)`**class MultiStringScanner**(`patterns`)Bases: `ScannerInterface`**property context:** `ContextInterface | None`**property layer_name:** `str | None`

search(*haystack*)

Return type

Generator[Tuple[int, bytes], None, None]

thread_safe = True

version = (0, 0, 0)

class **RegexScanner**(*pattern*, *flags*=*RegexFlag.DOTALL*)

Bases: *ScannerInterface*

A scanner that can be provided with a bytes-object regular expression pattern The scanner will scan all blocks for the regular expression and report the absolute offset of any finds

The default flags include DOTALL, since the searches are through binary data and the newline character should have no specific significance in such searches

property context: *ContextInterface* | None

property layer_name: str | None

thread_safe = True

version = (0, 0, 0)

Submodules

volatility3.framework.layers.scanners.multiregexp module

class **MultiRegexp**

Bases: *object*

Algorithm for multi-string matching.

add_pattern(*pattern*)

Return type

None

preprocess()

Return type

None

search(*haystack*)

Return type

Generator[Tuple[int, bytes], None, None]

Submodules

volatility3.framework.layers.avml module

Functions that read AVML files.

The user of the file doesn't have to worry about the compression, but random access is not allowed.

class **AVMLLayer**(*args, **kwargs)

Bases: *NonLinearlySegmentedLayer*

A Lime format TranslationLayer.

Lime is generally used to store physical memory images where there are large holes in the physical layer

Basic initializer that allows configurables to access their own config settings.

property **address_mask**: **int**

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property **config**: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property **config_path**: **str**

The configuration path on which this configurable lives.

property **context**: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

property **dependencies**: **List[str]**

Returns a list of the lower layers that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

None

classmethod **get_requirements**()

Returns a list of Requirement objects for this type of layer.

Return type

List[RequirementInterface]

is_valid(offset, length=1)

Returns whether the address offset can be translated to a valid address.

Return type`bool`**classmethod** `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**mapping**(`offset, length, ignore_errors=False`)

Returns a sorted iterable of (offset, length, mapped_offset, mapped_length, layer) mappings.

Return type`Iterable[Tuple[int, int, int, int, str]]`**property** `maximum_address: int`

Returns the maximum valid address of the space.

property `metadata: Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

property `minimum_address: int`

Returns the minimum valid address of the space.

property `name: str`

Returns the layer name.

read(`offset, length, pad=False`)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type`bytes`**scan**(`context, scanner, progress_callback=None, sections=None`)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (`ContextInterface`) – The context containing the data layer
- **scanner** (`ScannerInterface`) – The constructed Scanner object to be applied
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – Method that is called periodically during scanning to update progress
- **sections** (`Iterable[Tuple[int, int]]`) – A list of (start, size) tuples defining the portions of the layer to scan

Return type`Iterable[Any]`**Returns**

The output iterable from the scanner object having been run against the layer

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`

write(*offset, value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type`None`

class `AVMLStacker`

Bases: `StackerLayerInterface`

exclusion_list: `List[str] = []`

The list operating systems/first-level plugin hierarchy that should exclude this stacker

classmethod `stack(context, layer_name, progress_callback=None)`

Method to determine whether this builder can operate on the named layer. If so, modify the context appropriately.

Returns the name of any new layer stacked on top of this layer or None. The stacking is therefore strictly linear rather than tree driven.

Configuration options provided by the context are ignored, and defaults are to be used by this method to build a space where possible.

Parameters

- **context** (`ContextInterface`) – Context in which to construct the higher layer
- **layer_name** (`str`) – Name of the layer to stack on top of
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callback function to indicate progress through a scan (if one is necessary)

Return type`Optional[DataLayerInterface]`

stack_order = 10

The order in which to attempt stacking, the lower the earlier

classmethod `stacker_slow_warning()`

exception `SnappyException`

Bases: `VolatilityException`

add_note()

Exception.add_note(note) – add a note to the exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

uncompress(s)

Uncompress a snappy compressed string.

volatility3.framework.layers.cloudstorage module

volatility3.framework.layers.crash module

class WindowsCrashDump32Layer(context, config_path, name)

Bases: *SegmentedLayer*

A Windows crash format TranslationLayer. This TranslationLayer supports Microsoft complete memory dump files. It currently does not support kernel or small memory dump files.

Basic initializer that allows configurables to access their own config settings.

SIGNATURE = 1162297680

VALIDDDUMP = 1347245380

property address_mask: int

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

classmethod check_header(base_layer, offset=0)

Return type

Tuple[int, int]

property config: HierarchicalDict

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: str

The configuration path on which this configurable lives.

property context: ContextInterface

The context object that this configurable belongs to/configuration is stored in.

crashdump_json = 'crash'

property dependencies: List[str]

Returns a list of the lower layers that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

None

dump_header_name = `'_DUMP_HEADER'`

get_header()**Return type**

ObjectInterface

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

List[RequirementInterface]

get_summary_header()**Return type**

ObjectInterface

headerpages = `1`

is_valid(offset, length=1)

Returns whether the address offset can be translated to a valid address.

Return type

bool

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

mapping(offset, length, ignore_errors=False)

Returns a sorted iterable of (offset, length, mapped_offset, mapped_length, layer) mappings.

Return type

Iterable[Tuple[int, int, int, int, str]]

property maximum_address: *int*

Returns the maximum valid address of the space.

property metadata: `Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

property minimum_address: `int`

Returns the minimum valid address of the space.

property name: `str`

Returns the layer name.

provides = {'type': 'physical'}

read(*offset, length, pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

`bytes`

scan(*context, scanner, progress_callback=None, sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (`ContextInterface`) – The context containing the data layer
- **scanner** (`ScannerInterface`) – The constructed Scanner object to be applied
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – Method that is called periodically during scanning to update progress
- **sections** (`Iterable[Tuple[int, int]]`) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

`Iterable[Any]`

Returns

The output iterable from the scanner object having been run against the layer

supported_dumptypes = [1, 5]

translate(*offset, ignore_errors=False*)**Return type**

`Tuple[Optional[int], Optional[str]]`

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

write(*offset, value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

None

class WindowsCrashDump64Layer(*context, config_path, name*)

Bases: *WindowsCrashDump32Layer*

A Windows crash format TranslationLayer. This TranslationLayer supports Microsoft complete memory dump files. It currently does not support kernel or small memory dump files.

Basic initializer that allows configurables to access their own config settings.

SIGNATURE = 1162297680

VALIDDUMP = 875976004

property address_mask: *int*

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

classmethod check_header(*base_layer, offset=0*)

Return type

Tuple[int, int]

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

crashdump_json = 'crash64'

property dependencies: *List[str]*

Returns a list of the lower layers that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

None

dump_header_name = '_DUMP_HEADER64'

get_header()

Return type

ObjectInterface

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

List[RequirementInterface]

get_summary_header()

Return type

ObjectInterface

headerpages = 2

is_valid(offset, length=1)

Returns whether the address offset can be translated to a valid address.

Return type

bool

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

mapping(offset, length, ignore_errors=False)

Returns a sorted iterable of (offset, length, mapped_offset, mapped_length, layer) mappings.

Return type

Iterable[Tuple[int, int, int, int, str]]

property maximum_address: int

Returns the maximum valid address of the space.

property metadata: Mapping

Returns a ReadOnly copy of the metadata published by this layer.

property minimum_address: int

Returns the minimum valid address of the space.

property name: str

Returns the layer name.

provides = {'type': 'physical'}

read(*offset*, *length*, *pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

`bytes`

scan(*context*, *scanner*, *progress_callback=None*, *sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (*ContextInterface*) – The context containing the data layer
- **scanner** (*ScannerInterface*) – The constructed Scanner object to be applied
- **progress_callback** (*Optional*[*Callable*[[*float*, *str*], *None*]]) – Method that is called periodically during scanning to update progress
- **sections** (*Iterable*[*Tuple*[*int*, *int*]]) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

`Iterable[Any]`

Returns

The output iterable from the scanner object having been run against the layer

supported_dumptypes = [1, 5]

translate(*offset*, *ignore_errors=False*)

Return type

`Tuple[Optional[int], Optional[str]]`

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

write(*offset*, *value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

`None`

exception WindowsCrashDumpFormatException(*layer_name*, *args)

Bases: *LayerException*

Thrown when an error occurs with the underlying Crash file format.

add_note()

Exception.add_note(note) – add a note to the exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class WindowsCrashDumpStacker

Bases: *StackerLayerInterface*

exclusion_list: List[str] = []

The list operating systems/first-level plugin hierarchy that should exclude this stacker

classmethod stack(context, layer_name, progress_callback=None)

Method to determine whether this builder can operate on the named layer. If so, modify the context appropriately.

Returns the name of any new layer stacked on top of this layer or None. The stacking is therefore strictly linear rather than tree driven.

Configuration options provided by the context are ignored, and defaults are to be used by this method to build a space where possible.

Parameters

- **context** (*ContextInterface*) – Context in which to construct the higher layer
- **layer_name** (*str*) – Name of the layer to stack on top of
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callback function to indicate progress through a scan (if one is necessary)

Return type

Optional[DataLayerInterface]

stack_order = 11

The order in which to attempt stacking, the lower the earlier

classmethod stacker_slow_warning()

volatility3.framework.layers.elf module

class Elf64Layer(context, config_path, name)

Bases: *SegmentedLayer*

A layer that supports the Elf64 format as documented at: <http://ftp.openwatcom.org/devel/docs/elf-64-gen.pdf>

Basic initializer that allows configurables to access their own config settings.

ELF_CLASS = 2

MAGIC = 1179403647

property address_mask: int

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

property dependencies: *List[str]*

Returns a list of the lower layers that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

None

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

List[RequirementInterface]

is_valid(offset, length=1)

Returns whether the address offset can be translated to a valid address.

Return type

bool

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

mapping(*offset, length, ignore_errors=False*)

Returns a sorted iterable of (offset, length, mapped_offset, mapped_length, layer) mappings.

Return type

`Iterable[Tuple[int, int, int, int, str]]`

property maximum_address: `int`

Returns the maximum valid address of the space.

property metadata: `Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

property minimum_address: `int`

Returns the minimum valid address of the space.

property name: `str`

Returns the layer name.

read(*offset, length, pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

`bytes`

scan(*context, scanner, progress_callback=None, sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (`ContextInterface`) – The context containing the data layer
- **scanner** (`ScannerInterface`) – The constructed Scanner object to be applied
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – Method that is called periodically during scanning to update progress
- **sections** (`Iterable[Tuple[int, int]]`) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

`Iterable[Any]`

Returns

The output iterable from the scanner object having been run against the layer

translate(*offset, ignore_errors=False*)

Return type

`Tuple[Optional[int], Optional[str]]`

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`**write**(*offset, value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type`None`**class Elf64Stacker**

Bases: `StackerLayerInterface`

exclusion_list: `List[str] = []`

The list operating systems/first-level plugin hierarchy that should exclude this stacker

classmethod stack(*context, layer_name, progress_callback=None*)

Method to determine whether this builder can operate on the named layer. If so, modify the context appropriately.

Returns the name of any new layer stacked on top of this layer or None. The stacking is therefore strictly linear rather than tree driven.

Configuration options provided by the context are ignored, and defaults are to be used by this method to build a space where possible.

Parameters

- **context** (`ContextInterface`) – Context in which to construct the higher layer
- **layer_name** (`str`) – Name of the layer to stack on top of
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callback function to indicate progress through a scan (if one is necessary)

Return type`Optional[DataLayerInterface]`**stack_order** = 10

The order in which to attempt stacking, the lower the earlier

classmethod stacker_slow_warning()**exception ElfFormatException**(*layer_name, *args*)

Bases: `LayerException`

Thrown when an error occurs with the underlying ELF file format.

add_note()

Exception.add_note(note) – add a note to the exception

args**with_traceback**()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

volatility3.framework.layers.intel module**class Intel**(context, config_path, name, metadata=None)Bases: *LinearlyMappedLayer*

Translation Layer for the Intel IA32 memory mapping.

Basic initializer that allows configurables to access their own config settings.

property address_mask: *int*

Returns a mask which encapsulates all the active bits of an address for this layer.

bits_per_register = 32**build_configuration**()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***canonicalize**(addr)

Canonicalizes an address by performing an appropriate sign extension on the higher addresses

Return type*int***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

decanonicalize(addr)

Removes canonicalization to ensure an address fits within the correct range if it has been canonicalized

This will produce an address outside the range if the canonicalization is incorrect

Return type*int***property dependencies:** *List[str]*

Returns a list of the lower layer names that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type*None*

classmethod `get_requirements()`

Returns a list of Requirement objects for this type of layer.

Return type

`List[RequirementInterface]`

is_dirty(*offset*)

Returns whether the page at offset is marked dirty

Return type

`bool`

is_valid(*offset*, *length=1*)

Returns whether the address offset can be translated to a valid address.

Return type

`bool`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

mapping(*offset*, *length*, *ignore_errors=False*)

Returns a sorted iterable of (offset, sublength, mapped_offset, mapped_length, layer) mappings.

This allows translation layers to provide maps of contiguous regions in one layer

Return type

`Iterable[Tuple[int, int, int, int, str]]`

maximum_address = 4294967295

property metadata: `Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

minimum_address = 0

property name: `str`

Returns the layer name.

page_size = 4096

read(*offset*, *length*, *pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

`bytes`

scan(*context*, *scanner*, *progress_callback=None*, *sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (*ContextInterface*) – The context containing the data layer
- **scanner** (*ScannerInterface*) – The constructed Scanner object to be applied
- **progress_callback** (*Optional*[*Callable*[[*float*, *str*], *None*]]) – Method that is called periodically during scanning to update progress
- **sections** (*Iterable*[*Tuple*[*int*, *int*]]) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

Iterable[*Any*]

Returns

The output iterable from the scanner object having been run against the layer

```
structure = [('page directory', 10, False), ('page table', 10, True)]
```

translate(*offset*, *ignore_errors=False*)

Return type

Tuple[*Optional*[*int*], *Optional*[*str*]]

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[*str*, *RequirementInterface*]

write(*offset*, *value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

None

class Intel32e(*context*, *config_path*, *name*, *metadata=None*)

Bases: *Intel*

Class for handling 64-bit (32-bit extensions) for Intel architectures.

Basic initializer that allows configurables to access their own config settings.

property address_mask: *int*

Returns a mask which encapsulates all the active bits of an address for this layer.

bits_per_register = 64

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

canonicalize(addr)

Canonicalizes an address by performing an appropriate sign extension on the higher addresses

Return type

int

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

decanonicalize(addr)

Removes canonicalization to ensure an address fits within the correct range if it has been canonicalized

This will produce an address outside the range if the canonicalization is incorrect

Return type

int

property dependencies: *List[str]*

Returns a list of the lower layer names that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

None

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

List[RequirementInterface]

is_dirty(offset)

Returns whether the page at offset is marked dirty

Return type

bool

is_valid(offset, length=1)

Returns whether the address offset can be translated to a valid address.

Return type

bool

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

mapping(*offset, length, ignore_errors=False*)

Returns a sorted iterable of (offset, sublength, mapped_offset, mapped_length, layer) mappings.

This allows translation layers to provide maps of contiguous regions in one layer

Return type

Iterable[Tuple[int, int, int, int, str]]

maximum_address = 281474976710655

property metadata: *Mapping*

Returns a ReadOnly copy of the metadata published by this layer.

minimum_address = 0

property name: *str*

Returns the layer name.

page_size = 4096

read(*offset, length, pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

bytes

scan(*context, scanner, progress_callback=None, sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (*ContextInterface*) – The context containing the data layer
- **scanner** (*ScannerInterface*) – The constructed Scanner object to be applied
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – Method that is called periodically during scanning to update progress
- **sections** (*Iterable[Tuple[int, int]]*) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

Iterable[Any]

Returns

The output iterable from the scanner object having been run against the layer

```
structure = [('page map layer 4', 9, False), ('page directory pointer', 9, True),
('page directory', 9, True), ('page table', 9, True)]
```

```
translate(offset, ignore_errors=False)
```

Return type

`Tuple[Optional[int], Optional[str]]`

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

write(*offset, value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

`None`

class `IntelPAE(context, config_path, name, metadata=None)`

Bases: `Intel`

Class for handling Physical Address Extensions for Intel architectures.

Basic initializer that allows configurables to access their own config settings.

property `address_mask: int`

Returns a mask which encapsulates all the active bits of an address for this layer.

`bits_per_register = 32`

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

canonicalize(addr)

Canonicalizes an address by performing an appropriate sign extension on the higher addresses

Return type

`int`

property `config: HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

decanonicalize(*addr*)

Removes canonicalization to ensure an address fits within the correct range if it has been canonicalized

This will produce an address outside the range if the canonicalization is incorrect

Return type

`int`

property dependencies: `List[str]`

Returns a list of the lower layer names that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

`None`

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

`List[RequirementInterface]`

is_dirty(*offset*)

Returns whether the page at offset is marked dirty

Return type

`bool`

is_valid(*offset*, *length=1*)

Returns whether the address offset can be translated to a valid address.

Return type

`bool`

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

mapping(*offset, length, ignore_errors=False*)

Returns a sorted iterable of (offset, sublength, mapped_offset, mapped_length, layer) mappings.

This allows translation layers to provide maps of contiguous regions in one layer

Return type

`Iterable[Tuple[int, int, int, int, str]]`

maximum_address = 4294967295

property metadata: `Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

minimum_address = 0

property name: `str`

Returns the layer name.

page_size = 4096

read(*offset, length, pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

`bytes`

scan(*context, scanner, progress_callback=None, sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (`ContextInterface`) – The context containing the data layer
- **scanner** (`ScannerInterface`) – The constructed Scanner object to be applied
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – Method that is called periodically during scanning to update progress
- **sections** (`Iterable[Tuple[int, int]]`) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

`Iterable[Any]`

Returns

The output iterable from the scanner object having been run against the layer

structure = [('page directory pointer', 2, False), ('page directory', 9, True), ('page table', 9, True)]

translate(*offset, ignore_errors=False*)

Return type

`Tuple[Optional[int], Optional[str]]`

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```

unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))

```

Return type`Dict[str, RequirementInterface]`**write**(*offset, value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type`None`**class** `WindowsIntel`(*context, config_path, name, metadata=None*)

Bases: `WindowsMixin, Intel`

Basic initializer that allows configurables to access their own config settings.

property `address_mask`: `int`

Returns a mask which encapsulates all the active bits of an address for this layer.

bits_per_register = 32**build_configuration**()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too.

Return type`HierarchicalDict`**canonicalize**(*addr*)

Canonicalizes an address by performing an appropriate sign extension on the higher addresses.

Return type`int`**property** `config`: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property `config_path`: `str`

The configuration path on which this configurable lives.

property `context`: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

decanonicalize(*addr*)

Removes canonicalization to ensure an address fits within the correct range if it has been canonicalized.

This will produce an address outside the range if the canonicalization is incorrect.

Return type`int`**property** `dependencies`: `List[str]`

Returns a list of the lower layer names that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

`None`

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

`List[RequirementInterface]`

is_dirty(offset)

Returns whether the page at offset is marked dirty

Return type

`bool`

is_valid(offset, length=1)

Returns whether the address offset can be translated to a valid address.

Return type

`bool`

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

mapping(offset, length, ignore_errors=False)

Returns a sorted iterable of (offset, sublength, mapped_offset, mapped_length, layer) mappings.

This allows translation layers to provide maps of contiguous regions in one layer

Return type

`Iterable[Tuple[int, int, int, int, str]]`

maximum_address = 4294967295

property metadata: `Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

minimum_address = 0

property name: `str`

Returns the layer name.

page_size = 4096

read(*offset, length, pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

`bytes`

scan(*context, scanner, progress_callback=None, sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (*ContextInterface*) – The context containing the data layer
- **scanner** (*ScannerInterface*) – The constructed Scanner object to be applied
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – Method that is called periodically during scanning to update progress
- **sections** (*Iterable[Tuple[int, int]]*) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

`Iterable[Any]`

Returns

The output iterable from the scanner object having been run against the layer

structure = [('page directory', 10, False), ('page table', 10, True)]

translate(*offset, ignore_errors=False*)

Return type

`Tuple[Optional[int], Optional[str]]`

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

write(*offset, value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

`None`

class `WindowsIntel32e(context, config_path, name, metadata=None)`

Bases: `WindowsMixin`, `Intel32e`

Basic initializer that allows configurables to access their own config settings.

property `address_mask`: `int`

Returns a mask which encapsulates all the active bits of an address for this layer.

bits_per_register = 64

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

canonicalize(addr)

Canonicalizes an address by performing an appropriate sign extension on the higher addresses

Return type

`int`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

decanonicalize(addr)

Removes canonicalization to ensure an adress fits within the correct range if it has been canonicalized

This will produce an address outside the range if the canonicalization is incorrect

Return type

`int`

property dependencies: `List[str]`

Returns a list of the lower layer names that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

`None`

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

`List[RequirementInterface]`

is_dirty(*offset*)

Returns whether the page at offset is marked dirty

Return type

`bool`

is_valid(*offset*, *length=1*)

Returns whether the address offset can be translated to a valid address.

Return type

`bool`

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

mapping(*offset*, *length*, *ignore_errors=False*)

Returns a sorted iterable of (offset, sublength, mapped_offset, mapped_length, layer) mappings.

This allows translation layers to provide maps of contiguous regions in one layer

Return type

`Iterable[Tuple[int, int, int, int, str]]`

maximum_address = 281474976710655

property metadata: `Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

minimum_address = 0

property name: `str`

Returns the layer name.

page_size = 4096

read(*offset*, *length*, *pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

`bytes`

scan(*context*, *scanner*, *progress_callback=None*, *sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (*ContextInterface*) – The context containing the data layer
- **scanner** (*ScannerInterface*) – The constructed Scanner object to be applied
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – Method that is called periodically during scanning to update progress
- **sections** (*Iterable[Tuple[int, int]]*) – A list of (start, size) tuples defining the portions of the layer to scan

Return type*Iterable[Any]***Returns**

The output iterable from the scanner object having been run against the layer

```
structure = [('page map layer 4', 9, False), ('page directory pointer', 9, True),
('page directory', 9, True), ('page table', 9, True)]
```

```
translate(offset, ignore_errors=False)
```

Return type*Tuple[Optional[int], Optional[str]]*

```
classmethod unsatisfied(context, config_path)
```

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type*Dict[str, RequirementInterface]*

```
write(offset, value)
```

Writes a value at offset, distributing the writing across any underlying mapping.

Return type*None*

```
class WindowsIntelPAE(context, config_path, name, metadata=None)
```

Bases: *WindowsMixin*, *IntelPAE*

Basic initializer that allows configurables to access their own config settings.

```
property address_mask: int
```

Returns a mask which encapsulates all the active bits of an address for this layer.

```
bits_per_register = 32
```

```
build_configuration()
```

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict*

canonicalize(addr)

Canonicalizes an address by performing an appropriate sign extension on the higher addresses

Return type

`int`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

decanonicalize(addr)

Removes canonicalization to ensure an address fits within the correct range if it has been canonicalized

This will produce an address outside the range if the canonicalization is incorrect

Return type

`int`

property dependencies: `List[str]`

Returns a list of the lower layer names that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

`None`

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

`List[RequirementInterface]`

is_dirty(offset)

Returns whether the page at offset is marked dirty

Return type

`bool`

is_valid(offset, length=1)

Returns whether the address offset can be translated to a valid address.

Return type

`bool`

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration

- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

mapping(*offset, length, ignore_errors=False*)

Returns a sorted iterable of (offset, sublength, mapped_offset, mapped_length, layer) mappings.

This allows translation layers to provide maps of contiguous regions in one layer

Return type

Iterable[Tuple[int, int, int, int, str]]

maximum_address = 4294967295

property metadata: *Mapping*

Returns a ReadOnly copy of the metadata published by this layer.

minimum_address = 0

property name: *str*

Returns the layer name.

page_size = 4096

read(*offset, length, pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

bytes

scan(*context, scanner, progress_callback=None, sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (*ContextInterface*) – The context containing the data layer
- **scanner** (*ScannerInterface*) – The constructed Scanner object to be applied
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – Method that is called periodically during scanning to update progress
- **sections** (*Iterable[Tuple[int, int]]*) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

Iterable[Any]

Returns

The output iterable from the scanner object having been run against the layer

structure = [('page directory pointer', 2, False), ('page directory', 9, True), ('page table', 9, True)]

translate(*offset*, *ignore_errors=False*)

Return type

`Tuple[Optional[int], Optional[str]]`

classmethod **unsatisfied**(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

write(*offset*, *value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

`None`

class **WindowsMixin**(*context*, *config_path*, *name*, *metadata=None*)

Bases: `Intel`

Basic initializer that allows configurables to access their own config settings.

property **address_mask**: `int`

Returns a mask which encapsulates all the active bits of an address for this layer.

bits_per_register = 32

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

canonicalize(*addr*)

Canonicalizes an address by performing an appropriate sign extension on the higher addresses

Return type

`int`

property **config**: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property **config_path**: `str`

The configuration path on which this configurable lives.

property **context**: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

decanonicalize(addr)

Removes canonicalization to ensure an address fits within the correct range if it has been canonicalized

This will produce an address outside the range if the canonicalization is incorrect

Return type`int`**property dependencies:** `List[str]`

Returns a list of the lower layer names that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type`None`**classmethod get_requirements()**

Returns a list of Requirement objects for this type of layer.

Return type`List[RequirementInterface]`**is_dirty(offset)**

Returns whether the page at offset is marked dirty

Return type`bool`**is_valid(offset, length=1)**

Returns whether the address offset can be translated to a valid address.

Return type`bool`**classmethod make_subconfig(context, base_config_path, **kwargs)**

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**mapping(offset, length, ignore_errors=False)**

Returns a sorted iterable of (offset, sublength, mapped_offset, mapped_length, layer) mappings.

This allows translation layers to provide maps of contiguous regions in one layer

Return type`Iterable[Tuple[int, int, int, int, str]]``maximum_address = 4294967295``property metadata: Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

`minimum_address = 0``property name: str`

Returns the layer name.

`page_size = 4096``read(offset, length, pad=False)`

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type`bytes``scan(context, scanner, progress_callback=None, sections=None)`

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (`ContextInterface`) – The context containing the data layer
- **scanner** (`ScannerInterface`) – The constructed Scanner object to be applied
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – Method that is called periodically during scanning to update progress
- **sections** (`Iterable[Tuple[int, int]]`) – A list of (start, size) tuples defining the portions of the layer to scan

Return type`Iterable[Any]`**Returns**

The output iterable from the scanner object having been run against the layer

`structure = [('page directory', 10, False), ('page table', 10, True)]``translate(offset, ignore_errors=False)`**Return type**`Tuple[Optional[int], Optional[str]]``classmethod unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```

unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))

```

Return type`Dict[str, RequirementInterface]`

write(*offset*, *value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

`None`

volatility3.framework.layers.leechcore module

volatility3.framework.layers.lime module

exception `LineFormatException`(*layer_name*, **args*)

Bases: `LayerException`

Thrown when an error occurs with the underlying Lime file format.

add_note()

Exception.add_note(note) – add a note to the exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class `LimeLayer`(*context*, *config_path*, *name*)

Bases: `SegmentedLayer`

A Lime format TranslationLayer.

Lime is generally used to store physical memory images where there are large holes in the physical layer

Basic initializer that allows configurables to access their own config settings.

MAGIC = 1281969477

VERSION = 1

property `address_mask`: `int`

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property `config`: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property `config_path`: `str`

The configuration path on which this configurable lives.

property `context`: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

property dependencies: `List[str]`

Returns a list of the lower layers that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

`None`

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

`List[RequirementInterface]`

is_valid(offset, length=1)

Returns whether the address offset can be translated to a valid address.

Return type

`bool`

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

mapping(offset, length, ignore_errors=False)

Returns a sorted iterable of (offset, length, mapped_offset, mapped_length, layer) mappings.

Return type

`Iterable[Tuple[int, int, int, int, str]]`

property maximum_address: `int`

Returns the maximum valid address of the space.

property metadata: `Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

property minimum_address: `int`

Returns the minimum valid address of the space.

property name: `str`

Returns the layer name.

read(*offset*, *length*, *pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

`bytes`

scan(*context*, *scanner*, *progress_callback=None*, *sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (*ContextInterface*) – The context containing the data layer
- **scanner** (*ScannerInterface*) – The constructed Scanner object to be applied
- **progress_callback** (*Optional*[*Callable*[[*float*, *str*], *None*]]) – Method that is called periodically during scanning to update progress
- **sections** (*Iterable*[*Tuple*[*int*, *int*]]) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

`Iterable[Any]`

Returns

The output iterable from the scanner object having been run against the layer

translate(*offset*, *ignore_errors=False*)

Return type

`Tuple[Optional[int], Optional[str]]`

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

write(*offset*, *value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

`None`

class LimeStacker

Bases: *StackerLayerInterface*

exclusion_list: `List[str] = []`

The list operating systems/first-level plugin hierarchy that should exclude this stacker

classmethod `stack(context, layer_name, progress_callback=None)`

Method to determine whether this builder can operate on the named layer. If so, modify the context appropriately.

Returns the name of any new layer stacked on top of this layer or None. The stacking is therefore strictly linear rather than tree driven.

Configuration options provided by the context are ignored, and defaults are to be used by this method to build a space where possible.

Parameters

- **context** (*ContextInterface*) – Context in which to construct the higher layer
- **layer_name** (*str*) – Name of the layer to stack on top of
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callback function to indicate progress through a scan (if one is necessary)

Return type

Optional[DataLayerInterface]

stack_order = 10

The order in which to attempt stacking, the lower the earlier

classmethod `stacker_slow_warning()`

volatility3.framework.layers.linear module

class `LinearlyMappedLayer(context, config_path, name, metadata=None)`

Bases: *TranslationLayerInterface*

Class to differentiate Linearly Mapped layers (where $a \Rightarrow b$ implies that $a + c \Rightarrow b + c$)

Basic initializer that allows configurables to access their own config settings.

property `address_mask`: *int*

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property `config`: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property `config_path`: *str*

The configuration path on which this configurable lives.

property `context`: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

abstract property `dependencies`: *List[str]*

Returns a list of layer names that this layer translates onto.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

`None`

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

`List[RequirementInterface]`

abstract is_valid(offset, length=1)

Returns a boolean based on whether the entire chunk of data (from offset to length) is valid or not.

Parameters

- **offset** (`int`) – The address to start determining whether bytes are readable/valid
- **length** (`int`) – The number of bytes from offset of which to test the validity

Return type

`bool`

Returns

Whether the bytes are valid and accessible

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

abstract mapping(offset, length, ignore_errors=False)

Returns a sorted iterable of (offset, sublength, mapped_offset, mapped_length, layer) mappings.

ignore_errors will provide all available maps with gaps, but their total length may not add up to the requested length This allows translation layers to provide maps of contiguous regions in one layer

Return type

`Iterable[Tuple[int, int, int, int, str]]`

abstract property maximum_address: int

Returns the maximum valid address of the space.

property metadata: Mapping

Returns a ReadOnly copy of the metadata published by this layer.

abstract property minimum_address: `int`

Returns the minimum valid address of the space.

property name: `str`

Returns the layer name.

read(*offset, length, pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

`bytes`

scan(*context, scanner, progress_callback=None, sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (*ContextInterface*) – The context containing the data layer
- **scanner** (*ScannerInterface*) – The constructed Scanner object to be applied
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – Method that is called periodically during scanning to update progress
- **sections** (*Iterable[Tuple[int, int]]*) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

`Iterable[Any]`

Returns

The output iterable from the scanner object having been run against the layer

translate(*offset, ignore_errors=False*)

Return type

`Tuple[Optional[int], Optional[str]]`

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

write(*offset, value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

`None`

volatility3.framework.layers.msf module**exception PDBFormatException**(*layer_name, *args*)Bases: *LayerException*

Thrown when an error occurs with the underlying MSF file format.

add_note()

Exception.add_note(note) – add a note to the exception

args**with_traceback()**

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class PdbMSFStream(*context, config_path, name, metadata=None*)Bases: *LinearlyMappedLayer*

Basic initializer that allows configurables to access their own config settings.

property address_mask: *int*

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

property dependencies: *List[str]*

Returns a list of layer names that this layer translates onto.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type*None***classmethod get_requirements()**

Returns a list of Requirement objects for this type of layer.

Return type*List[RequirementInterface]*

is_valid(*offset*, *length=1*)

Returns a boolean based on whether the entire chunk of data (from offset to length) is valid or not.

Parameters

- **offset** (*int*) – The address to start determining whether bytes are readable/valid
- **length** (*int*) – The number of bytes from offset of which to test the validity

Return type

bool

Returns

Whether the bytes are valid and accessible

classmethod **make_subconfig**(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

mapping(*offset*, *length*, *ignore_errors=False*)

Returns a sorted iterable of (offset, sublength, mapped_offset, mapped_length, layer) mappings.

ignore_errors will provide all available maps with gaps, but their total length may not add up to the requested length This allows translation layers to provide maps of contiguous regions in one layer

Return type

Iterable[Tuple[int, int, int, int, str]]

property **maximum_address**: *int*

Returns the maximum valid address of the space.

property **metadata**: *Mapping*

Returns a ReadOnly copy of the metadata published by this layer.

property **minimum_address**: *int*

Returns the minimum valid address of the space.

property **name**: *str*

Returns the layer name.

property **pdb_symbol_table**: *str* | *None*

read(*offset*, *length*, *pad=False*)

Reads an offset for length bytes and returns ‘bytes’ (not ‘str’) of length size.

Return type

bytes

scan(*context*, *scanner*, *progress_callback*=None, *sections*=None)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (*ContextInterface*) – The context containing the data layer
- **scanner** (*ScannerInterface*) – The constructed Scanner object to be applied
- **progress_callback** (*Optional*[*Callable*[[*float*, *str*], *None*]]) – Method that is called periodically during scanning to update progress
- **sections** (*Iterable*[*Tuple*[*int*, *int*]]) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

Iterable[*Any*]

Returns

The output iterable from the scanner object having been run against the layer

translate(*offset*, *ignore_errors*=False)

Return type

Tuple[*Optional*[*int*], *Optional*[*str*]]

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[*str*, *RequirementInterface*]

write(*offset*, *value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

None

class PdbMultiStreamFormat(*context*, *config_path*, *name*, *metadata*=None)

Bases: *LinearlyMappedLayer*

Basic initializer that allows configurables to access their own config settings.

property address_mask: *int*

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

create_stream_from_pages(*stream_name, maximum_size, pages*)**Return type***str***property dependencies:** *List[str]*

Returns a list of the lower layers that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type*None***classmethod get_requirements()**

Returns a list of Requirement objects for this type of layer.

Return type*List[RequirementInterface]***get_stream**(*index*)**Return type***Optional[PdbMSFStream]***is_valid**(*offset, length=1*)

Returns a boolean based on whether the entire chunk of data (from offset to length) is valid or not.

Parameters

- **offset** (*int*) – The address to start determining whether bytes are readable/valid
- **length** (*int*) – The number of bytes from offset of which to test the validity

Return type*bool***Returns**

Whether the bytes are valid and accessible

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration

- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

mapping(*offset, length, ignore_errors=False*)

Returns a sorted iterable of (offset, sublength, mapped_offset, mapped_length, layer) mappings.

ignore_errors will provide all available maps with gaps, but their total length may not add up to the requested length This allows translation layers to provide maps of contiguous regions in one layer

Return type

Iterable[Tuple[int, int, int, int, str]]

property maximum_address: *int*

Returns the maximum valid address of the space.

property metadata: *Mapping*

Returns a ReadOnly copy of the metadata published by this layer.

property minimum_address: *int*

Returns the minimum valid address of the space.

property name: *str*

Returns the layer name.

property page_size

property pdb_symbol_table: *str*

read(*offset, length, pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

bytes

read_streams()

scan(*context, scanner, progress_callback=None, sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (*ContextInterface*) – The context containing the data layer
- **scanner** (*ScannerInterface*) – The constructed Scanner object to be applied
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – Method that is called periodically during scanning to update progress
- **sections** (*Iterable[Tuple[int, int]]*) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

Iterable[Any]

Returns

The output iterable from the scanner object having been run against the layer

translate(*offset*, *ignore_errors=False*)

Return type

`Tuple[Optional[int], Optional[str]]`

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

write(*offset*, *value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

`None`

volatility3.framework.layers.physical module

class BufferDataLayer(*context*, *config_path*, *name*, *buffer*, *metadata=None*)

Bases: `DataLayerInterface`

A DataLayer class backed by a buffer in memory, designed for testing and swift data access.

Basic initializer that allows configurables to access their own config settings.

property address_mask: `int`

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

property dependencies: `List[str]`

A list of other layer names required by this layer.

Note: DataLayers must never define other layers

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

`None`

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

`List[RequirementInterface]`

is_valid(offset, length=1)

Returns whether the offset is valid or not.

Return type

`bool`

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property maximum_address: `int`

Returns the largest available address in the space.

property metadata: `Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

property minimum_address: `int`

Returns the smallest available address in the space.

property name: `str`

Returns the layer name.

read(*address*, *length*, *pad=False*)

Reads the data from the buffer.

Return type

`bytes`

scan(*context*, *scanner*, *progress_callback=None*, *sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (*ContextInterface*) – The context containing the data layer
- **scanner** (*ScannerInterface*) – The constructed Scanner object to be applied
- **progress_callback** (*Optional*[*Callable*[[*float*, *str*], *None*]]) – Method that is called periodically during scanning to update progress
- **sections** (*Iterable*[*Tuple*[*int*, *int*]]) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

`Iterable[Any]`

Returns

The output iterable from the scanner object having been run against the layer

classmethod **unsatisfied**(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

write(*address*, *data*)

Writes the data from to the buffer.

class **DummyLock**

Bases: *object*

class **FileLayer**(*context*, *config_path*, *name*, *metadata=None*)

Bases: *DataLayerInterface*

a DataLayer backed by a file on the filesystem.

Basic initializer that allows configurables to access their own config settings.

property **address_mask**: *int*

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

property dependencies: *List[str]*

A list of other layer names required by this layer.

Note: DataLayers must never define other layers

destroy()

Closes the file handle.

Return type

None

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

List[RequirementInterface]

is_valid(offset, length=1)

Returns whether the offset is valid or not.

Return type

bool

property location: *str*

Returns the location on which this Layer abstracts.

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property maximum_address:** `int`

Returns the largest available address in the space.

property metadata: `Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

property minimum_address: `int`

Returns the smallest available address in the space.

property name: `str`

Returns the layer name.

read(*offset, length, pad=False*)

Reads from the file at offset for length.

Return type`bytes`**scan**(*context, scanner, progress_callback=None, sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (`ContextInterface`) – The context containing the data layer
- **scanner** (`ScannerInterface`) – The constructed Scanner object to be applied
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – Method that is called periodically during scanning to update progress
- **sections** (`Iterable[Tuple[int, int]]`) – A list of (start, size) tuples defining the portions of the layer to scan

Return type`Iterable[Any]`**Returns**

The output iterable from the scanner object having been run against the layer

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```

unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))

```

Return type`Dict[str, RequirementInterface]`**write**(*offset, data*)

Writes to the file.

This will technically allow writes beyond the extent of the file

Return type`None`**volatility3.framework.layers.qemu module****class QemuStacker**Bases: `StackerLayerInterface`**exclusion_list:** `List[str] = []`

The list operating systems/first-level plugin hierarchy that should exclude this stacker

classmethod stack(*context, layer_name, progress_callback=None*)

Method to determine whether this builder can operate on the named layer. If so, modify the context appropriately.

Returns the name of any new layer stacked on top of this layer or `None`. The stacking is therefore strictly linear rather than tree driven.

Configuration options provided by the context are ignored, and defaults are to be used by this method to build a space where possible.

Parameters

- **context** (`ContextInterface`) – Context in which to construct the higher layer
- **layer_name** (`str`) – Name of the layer to stack on top of
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callback function to indicate progress through a scan (if one is necessary)

Return type`Optional[DataLayerInterface]`**stack_order** = 10

The order in which to attempt stacking, the lower the earlier

classmethod stacker_slow_warning()**class QemuSuspendLayer**(*context, config_path, name, metadata=None*)Bases: `NonLinearlySegmentedLayer`

A Qemu suspend-to-disk translation layer.

Basic initializer that allows configurables to access their own config settings.

HASH_PTE_SIZE_64 = 16**QEVm_CONFIGURATION** = 7**QEVm_EOF** = 0**QEVm_SECTION_END** = 3**QEVm_SECTION_FOOTER** = 126**QEVm_SECTION_FULL** = 4**QEVm_SECTION_PART** = 2**QEVm_SECTION_START** = 1

QEVMSUBSECTION = 5

QEVMSUBDESCRIPTION = 6

SEGMENT_FLAG_COMPRESS = 2

SEGMENT_FLAG_CONTINUE = 32

SEGMENT_FLAG_EOS = 16

SEGMENT_FLAG_HOOK = 128

SEGMENT_FLAG_MEM_SIZE = 4

SEGMENT_FLAG_PAGE = 8

SEGMENT_FLAG_XBZRLE = 64

property address_mask: `int`

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

property dependencies: `List[str]`

Returns a list of the lower layers that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

`None`

distro_re = `'(\\w+[\\d{1,2}\\.]*)'`

extract_data(*index, name, version_id*)

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

List[RequirementInterface]

is_valid(*offset*, *length=1*)

Returns whether the address offset can be translated to a valid address.

Return type

`bool`

classmethod **make_subconfig**(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

mapping(*offset*, *length*, *ignore_errors=False*)

Returns a sorted iterable of (offset, length, mapped_offset, mapped_length, layer) mappings.

Return type

`Iterable[Tuple[int, int, int, int, str]]`

property **maximum_address**: `int`

Returns the maximum valid address of the space.

property **metadata**: `Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

property **minimum_address**: `int`

Returns the minimum valid address of the space.

property **name**: `str`

Returns the layer name.

```
pci_hole_table = {re.compile('^pc-i440fx-([23456789]|\\d\\d+\\.\\.\\.\\d$'):  
(3758096384, 3221225472, 4294967296), re.compile('^pc-i440fx-[01]\\.\\.\\.\\d$'):  
(3758096384, 3758096384, 4294967296), re.compile('^pc-q35-\\d\\.\\.\\.\\d$'):  
(2952790016, 2147483648, 4294967296), re.compile('^microvm$'):  
(3221225472, 3221225472, 4294967296), re.compile('^xen$'):  
(4026531840, 4026531840, 4294967296),  
re.compile('^pc-i440fx-(\\w+[\\d{1,2}\\.]*)$'):  
(3758096384, 3221225472, 4294967296), re.compile('^pc-q35-(\\w+[\\d{1,2}\\.]*)$'):  
(2952790016, 2147483648, 4294967296)}
```

read(*offset*, *length*, *pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

`bytes`

scan(*context*, *scanner*, *progress_callback=None*, *sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (*ContextInterface*) – The context containing the data layer
- **scanner** (*ScannerInterface*) – The constructed Scanner object to be applied
- **progress_callback** (*Optional*[*Callable*[[*float*, *str*], *None*]]) – Method that is called periodically during scanning to update progress
- **sections** (*Iterable*[*Tuple*[*int*, *int*]]) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

Iterable[*Any*]

Returns

The output iterable from the scanner object having been run against the layer

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[*str*, *RequirementInterface*]

write(*offset*, *value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

None

volatility3.framework.layers.registry module

exception RegistryFormatException(*layer_name*, **args*)

Bases: *LayerException*

Thrown when an error occurs with the underlying Registry file format.

add_note()

Exception.add_note(note) – add a note to the exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class RegistryHive(*context, config_path, name, metadata=None*)

Bases: *LinearlyMappedLayer*

Basic initializer that allows configurables to access their own config settings.

property address_mask: *int*

Return a mask that allows for the volatile bit to be set.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

property dependencies: *List[str]*

Returns a list of layer names that this layer translates onto.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

None

get_cell(*cell_offset*)

Returns the appropriate Cell value for a cell offset.

Return type

StructType

get_key(*key, return_list=False*)

Gets a specific registry key by key path.

return_list specifies whether the return result will be a single node (default) or a list of nodes from root to the current node (if return_list is true).

Return type

Union[List[StructType], StructType]

get_name()

Return type

str

get_node(*cell_offset*)

Returns the appropriate Node, interpreted from the Cell based on its Signature.

Return type

StructType

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

List[RequirementInterface]

property hive_offset: *int*

is_valid(*offset*, *length=1*)

Returns a boolean based on whether the offset is valid or not.

Return type

bool

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

mapping(*offset*, *length*, *ignore_errors=False*)

Returns a sorted iterable of (offset, sublength, mapped_offset, mapped_length, layer) mappings.

ignore_errors will provide all available maps with gaps, but their total length may not add up to the requested length This allows translation layers to provide maps of contiguous regions in one layer

Return type

Iterable[Tuple[int, int, int, int, str]]

property maximum_address: *int*

Returns the maximum valid address of the space.

property metadata: *Mapping*

Returns a ReadOnly copy of the metadata published by this layer.

property minimum_address: *int*

Returns the minimum valid address of the space.

property name: *str*

Returns the layer name.

read(*offset*, *length*, *pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

`bytes`

property root_cell_offset: `int`

Returns the offset for the root cell in this hive.

scan(*context*, *scanner*, *progress_callback=None*, *sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (`ContextInterface`) – The context containing the data layer
- **scanner** (`ScannerInterface`) – The constructed Scanner object to be applied
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – Method that is called periodically during scanning to update progress
- **sections** (`Iterable[Tuple[int, int]]`) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

`Iterable[Any]`

Returns

The output iterable from the scanner object having been run against the layer

translate(*offset*, *ignore_errors=False*)

Return type

`Tuple[Optional[int], Optional[str]]`

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

visit_nodes(*visitor*, *node=None*)

Applies a callable (visitor) to all nodes within the registry tree from a given node.

Return type

`None`

write(*offset*, *value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

`None`

exception RegistryInvalidIndex(*layer_name*, **args*)

Bases: *LayerException*

Thrown when an index that doesn't exist or can't be found occurs.

add_note()

Exception.add_note(note) – add a note to the exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

volatility3.framework.layers.resources module

class JarHandler

Bases: *VolatilityHandler*

Handles the jar scheme for URIs.

Reference used for the schema syntax: <http://docs.netkernel.org/book/view/book:mod:reference/doc:layer1:schemes:jar>

Actual reference (found from <https://www.w3.org/wiki/UriSchemes/jar>) seemed not to return: <http://developer.java.sun.com/developer/onlineTraining/protocolhandlers/>

add_parent(*parent*)

close()

static default_open(*req*)

Handles the request if it's the jar scheme.

Return type

Optional[*Any*]

handler_order = 500

classmethod non_cached_schemes()

Return type

List[*str*]

class OfflineHandler

Bases: *VolatilityHandler*

add_parent(*parent*)

close()

static default_open(*req*)

Return type

Optional[*Any*]

handler_order = 500

classmethod `non_cached_schemes()`

Return type

`List[str]`

class `ResourceAccessor`(*progress_callback=None, context=None, enable_cache=True*)

Bases: `object`

Object for opening URLs as files (downloading locally first if necessary)

Creates a resource accessor.

Note: context is an SSL context, not a volatility context

list_handlers = `True`

open(*url, mode='rb'*)

Returns a file-like object for a particular URL opened in mode.

If the file is remote, it will be downloaded and locally cached

Return type

`Any`

uses_cache(*url*)

Determines whether a URLs contents should be cached

Return type

`bool`

class `VolatilityHandler`

Bases: `BaseHandler`

add_parent(*parent*)

close()

handler_order = `500`

classmethod `non_cached_schemes()`

Return type

`List[str]`

cascadeCloseFile(*new_fp, original_fp*)

Really horrible solution for ensuring files aren't left open

Parameters

- **new_fp** (`IO[bytes]`) – The file pointer constructed based on the original file pointer
- **original_fp** (`IO[bytes]`) – The original file pointer that should be closed when the new file pointer is closed, but isn't

Return type

`IO[bytes]`

volatility3.framework.layers.segmented module

class NonLinearlySegmentedLayer(*context, config_path, name, metadata=None*)

Bases: *TranslationLayerInterface*

A class to handle a single run-based layer-to-layer mapping.

In the documentation “mapped address” or “mapped offset” refers to an offset once it has been mapped to the underlying layer

Basic initializer that allows configurables to access their own config settings.

property address_mask: *int*

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

property dependencies: *List[str]*

Returns a list of the lower layers that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

None

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

List[RequirementInterface]

is_valid(*offset, length=1*)

Returns whether the address offset can be translated to a valid address.

Return type

bool

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

mapping(*offset, length, ignore_errors=False*)

Returns a sorted iterable of (offset, length, mapped_offset, mapped_length, layer) mappings.

Return type

Iterable[Tuple[int, int, int, int, str]]

property maximum_address: *int*

Returns the maximum valid address of the space.

property metadata: *Mapping*

Returns a ReadOnly copy of the metadata published by this layer.

property minimum_address: *int*

Returns the minimum valid address of the space.

property name: *str*

Returns the layer name.

read(*offset, length, pad=False*)

Reads an offset for length bytes and returns ‘bytes’ (not ‘str’) of length size.

Return type

bytes

scan(*context, scanner, progress_callback=None, sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (*ContextInterface*) – The context containing the data layer
- **scanner** (*ScannerInterface*) – The constructed Scanner object to be applied
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – Method that is called periodically during scanning to update progress
- **sections** (*Iterable[Tuple[int, int]]*) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

Iterable[Any]

Returns

The output iterable from the scanner object having been run against the layer

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

write(offset, value)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

`None`

class `SegmentedLayer(context, config_path, name, metadata=None)`

Bases: `NonLinearlySegmentedLayer`, `LinearlyMappedLayer`

Basic initializer that allows configurables to access their own config settings.

property `address_mask: int`

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property `config: HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property `config_path: str`

The configuration path on which this configurable lives.

property `context: ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

property `dependencies: List[str]`

Returns a list of the lower layers that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

`None`

classmethod `get_requirements()`

Returns a list of Requirement objects for this type of layer.

Return type

`List[RequirementInterface]`

is_valid(*offset*, *length=1*)

Returns whether the address offset can be translated to a valid address.

Return type

`bool`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

mapping(*offset*, *length*, *ignore_errors=False*)

Returns a sorted iterable of (offset, length, mapped_offset, mapped_length, layer) mappings.

Return type

`Iterable[Tuple[int, int, int, int, str]]`

property `maximum_address: int`

Returns the maximum valid address of the space.

property `metadata: Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

property `minimum_address: int`

Returns the minimum valid address of the space.

property `name: str`

Returns the layer name.

read(*offset*, *length*, *pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

`bytes`

scan(*context*, *scanner*, *progress_callback=None*, *sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (`ContextInterface`) – The context containing the data layer

- **scanner** (*ScannerInterface*) – The constructed Scanner object to be applied
- **progress_callback** (*Optional*[*Callable*[[*float*, *str*], *None*]]) – Method that is called periodically during scanning to update progress
- **sections** (*Iterable*[*Tuple*[*int*, *int*]]) – A list of (start, size) tuples defining the portions of the layer to scan

Return type*Iterable*[*Any*]**Returns**

The output iterable from the scanner object having been run against the layer

translate(*offset*, *ignore_errors=False*)**Return type***Tuple*[*Optional*[*int*], *Optional*[*str*]]**classmethod** **unsatisfied**(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```

unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))

```

Return type*Dict*[*str*, *RequirementInterface*]**write**(*offset*, *value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type*None***volatility3.framework.layers.vmware module****exception** **VmwareFormatException**(*layer_name*, **args*)Bases: *LayerException*

Thrown when an error occurs with the underlying VMware vmem file format.

add_note()

Exception.add_note(note) – add a note to the exception

args**with_traceback**()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class **VmwareLayer**(*context*, *config_path*, *name*, *metadata=None*)Bases: *SegmentedLayer*

Basic initializer that allows configurables to access their own config settings.

property address_mask: `int`

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

property dependencies: `List[str]`

Returns a list of the lower layers that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

`None`

classmethod get_requirements()

This vmware translation layer always requires a separate metadata layer.

Return type

`List[RequirementInterface]`

group_structure = `'64sQQ'`

header_structure = `'<4sII'`

is_valid(offset, length=1)

Returns whether the address offset can be translated to a valid address.

Return type

`bool`

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

mapping(*offset, length, ignore_errors=False*)

Returns a sorted iterable of (offset, length, mapped_offset, mapped_length, layer) mappings.

Return type

`Iterable[Tuple[int, int, int, int, str]]`

property maximum_address: `int`

Returns the maximum valid address of the space.

property metadata: `Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

property minimum_address: `int`

Returns the minimum valid address of the space.

property name: `str`

Returns the layer name.

read(*offset, length, pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

`bytes`

scan(*context, scanner, progress_callback=None, sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (`ContextInterface`) – The context containing the data layer
- **scanner** (`ScannerInterface`) – The constructed Scanner object to be applied
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – Method that is called periodically during scanning to update progress
- **sections** (`Iterable[Tuple[int, int]]`) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

`Iterable[Any]`

Returns

The output iterable from the scanner object having been run against the layer

translate(*offset, ignore_errors=False*)

Return type

`Tuple[Optional[int], Optional[str]]`

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

write(offset, value)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

`None`

class VmwareStacker

Bases: `StackerLayerInterface`

exclusion_list: `List[str] = []`

The list operating systems/first-level plugin hierarchy that should exclude this stacker

classmethod stack(context, layer_name, progress_callback=None)

Attempt to stack this based on the starting information.

Return type

`Optional[DataLayerInterface]`

stack_order = 20

The order in which to attempt stacking, the lower the earlier

classmethod stacker_slow_warning()

volatility3.framework.layers.xen module

class XenCoreDumpLayer(context, config_path, name)

Bases: `Elf64Layer`

A layer that supports the Xen Dump-Core format as documented at: <https://xenbits.xen.org/docs/4.6-testing/misc/dump-core-format.txt>

Basic initializer that allows configurables to access their own config settings.

ELF_CLASS = 2

MAGIC = 1179403647

property address_mask: `int`

Returns a mask which encapsulates all the active bits of an address for this layer.

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

property dependencies: *List[str]*

Returns a list of the lower layers that this layer is dependent upon.

destroy()

Causes a DataLayer to close any open handles, etc.

Systems that make use of Data Layers should call destroy when they are done with them. This will close all handles, and make the object unreadable (exceptions will be thrown using a DataLayer after destruction)

Return type

None

classmethod get_requirements()

Returns a list of Requirement objects for this type of layer.

Return type

List[RequirementInterface]

is_valid(offset, length=1)

Returns whether the address offset can be translated to a valid address.

Return type

bool

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

mapping(offset, length, ignore_errors=False)

Returns a sorted iterable of (offset, length, mapped_offset, mapped_length, layer) mappings.

Return type

Iterable[Tuple[int, int, int, int, str]]

property maximum_address: *int*

Returns the maximum valid address of the space.

property metadata: `Mapping`

Returns a ReadOnly copy of the metadata published by this layer.

property minimum_address: `int`

Returns the minimum valid address of the space.

property name: `str`

Returns the layer name.

read(*offset, length, pad=False*)

Reads an offset for length bytes and returns 'bytes' (not 'str') of length size.

Return type

`bytes`

scan(*context, scanner, progress_callback=None, sections=None*)

Scans a Translation layer by chunk.

Note: this will skip missing/unmappable chunks of memory

Parameters

- **context** (`ContextInterface`) – The context containing the data layer
- **scanner** (`ScannerInterface`) – The constructed Scanner object to be applied
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – Method that is called periodically during scanning to update progress
- **sections** (`Iterable[Tuple[int, int]]`) – A list of (start, size) tuples defining the portions of the layer to scan

Return type

`Iterable[Any]`

Returns

The output iterable from the scanner object having been run against the layer

translate(*offset, ignore_errors=False*)

Return type

`Tuple[Optional[int], Optional[str]]`

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

write(*offset, value*)

Writes a value at offset, distributing the writing across any underlying mapping.

Return type

`None`

class XenCoreDumpStackerBases: *Elf64Stacker***exclusion_list**: List[str] = []

The list operating systems/first-level plugin hierarchy that should exclude this stacker

classmethod stack(context, layer_name, progress_callback=None)

Method to determine whether this builder can operate on the named layer. If so, modify the context appropriately.

Returns the name of any new layer stacked on top of this layer or None. The stacking is therefore strictly linear rather than tree driven.

Configuration options provided by the context are ignored, and defaults are to be used by this method to build a space where possible.

Parameters

- **context** (*ContextInterface*) – Context in which to construct the higher layer
- **layer_name** (str) – Name of the layer to stack on top of
- **progress_callback** (Optional[Callable[[float, str], None]]) – A callback function to indicate progress through a scan (if one is necessary)

Return typeOptional[*DataLayerInterface*]**stack_order** = 10

The order in which to attempt stacking, the lower the earlier

classmethod stacker_slow_warning()**volatility3.framework.objects package****class AggregateType**(context, type_name, object_info, size, members)Bases: *ObjectInterface*

Object which can contain members that are other objects.

Keep the number of methods in this class low or very specific, since each one could overload a valid member.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (str) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxyBases: *VolTemplateProxy***classmethod child_template**(template, child)

Returns the template of a child to its parent.

Return type*Template*

classmethod **children**(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod **has_member**(*template*, *member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

classmethod **relative_child_offset**(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod **replace_child**(*template*, *old_child*, *new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod **size**(*template*)

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the *table_name* is not valid within the object's context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class Array(*context, type_name, object_info, count=0, subtype=None*)

Bases: *ObjectInterface, Sequence*

Object which can contain a fixed number of an object type.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of the child member.

Return type

Template

classmethod children(*template*)

Returns the children of the template.

Return type

List[Template]

abstract classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset from the head of the parent data to the child member.

Return type

int

classmethod `replace_child(template, old_child, new_child)`

Substitutes the old_child for the new_child.

Return type

`None`

classmethod `size(template)`

Returns the size of the array, based on the count and the subtype.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

property count: `int`

Returns the count dynamically.

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Parameters

member_name (`str`) – Name to test whether a member exists within the type structure

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

index(*value*[, *start*[, *stop*]]) → integer -- return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

property *vol*: [*ReadOnlyMapping*](#)

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

Return type

[*None*](#)

class [*BitField*](#)(*context*, *type_name*, *object_info*, *base_type*, *start_bit*=0, *end_bit*=0)

Bases: [*ObjectInterface*](#), [*int*](#)

Object containing a field which is made up of bits rather than whole bytes.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** ([*ContextInterface*](#)) – The context associated with the object
- **type_name** ([*str*](#)) – The name of the type structure for the object
- **object_info** ([*ObjectInformation*](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class [*VolTemplateProxy*](#)

Bases: [*VolTemplateProxy*](#)

abstract classmethod [*child_template*](#)(*template*, *child*)

Returns the template of the child member from the parent.

Return type

[*Template*](#)

classmethod [*children*](#)(*template*)

Returns the children of the template.

Return type

[*List\[Template\]*](#)

abstract classmethod [*has_member*](#)(*template*, *member_name*)

Returns whether the object would contain a member called member_name.

Return type

[*bool*](#)

abstract classmethod [*relative_child_offset*](#)(*template*, *child*)

Returns the relative offset from the head of the parent data to the child member.

Return type

[*int*](#)

classmethod [*replace_child*](#)(*template*, *old_child*, *new_child*)

Substitutes the old_child for the new_child.

Return type

[*None*](#)

classmethod `size(template)`

Returns the size of the template object.

Return type

`int`

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

cast(new_type_name, **additional)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Parameters

member_name (`str`) – Name to test whether a member exists within the type structure

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

imag

the imaginary part of a complex number

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(length=1, byteorder='big', *, signed=False)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

class Boolean(*context, type_name, object_info, data_format*)

Bases: *PrimitiveObject, int*

Primitive Object that handles boolean types.

Constructs an Object adhering to the `ObjectInterface`.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

abstract classmethod child_template(*template, child*)

Returns the template of the child member from the parent.

Return type

Template

abstract classmethod children(*template*)

Returns the children of the template.

Return type

List[Template]

abstract classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called `member_name`.

Return type

bool

abstract classmethod relative_child_offset(*template, child*)

Returns the relative offset from the head of the parent data to the child member.

Return type

int

abstract classmethod `replace_child(template, old_child, new_child)`

Substitutes the `old_child` for the `new_child`.

Return type

`None`

classmethod `size(template)`

Returns the size of the templated object.

Return type

`int`

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

cast(new_type_name, **additional)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Parameters

member_name (`str`) – Name to test whether a member exists within the type structure

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

imag

the imaginary part of a complex number

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If `byteorder` is 'big', the most significant byte is at the beginning of the byte array. If `byteorder` is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If `signed` is `False` and a negative integer is given, an `OverflowError` is raised.

property vol: [*ReadOnlyMapping*](#)

Returns the volatility specific object information.

write(*value*)

Writes the object into the layer of the context at the current offset.

Return type

[*ObjectInterface*](#)

class Bytes(*context, type_name, object_info, length=1*)

Bases: [*PrimitiveObject*](#), `bytes`

Primitive Object that handles specific series of bytes.

Constructs an Object adhering to the `ObjectInterface`.

Parameters

- **context** ([*ContextInterface*](#)) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** ([*ObjectInformation*](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: [*VolTemplateProxy*](#)

abstract classmethod child_template(*template, child*)

Returns the template of the child member from the parent.

Return type

[*Template*](#)

abstract classmethod children(*template*)

Returns the children of the template.

Return type

[*List\[Template\]*](#)

abstract classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called `member_name`.

Return type`bool`**abstract classmethod** `relative_child_offset(template, child)`

Returns the relative offset from the head of the parent data to the child member.

Return type`int`**abstract classmethod** `replace_child(template, old_child, new_child)`

Substitutes the old_child for the new_child.

Return type`None`**classmethod** `size(template)`

Returns the size of the template object.

Return type`int`**capitalize()** → copy of B

Return a copy of B with only its first character capitalized (ASCII) and the rest lower-cased.

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

center(*width*, *fillchar=b' ', /*)

Return a centered string of length width.

Padding is done using the specified fill character.

count(*sub*[, *start*[, *end*]]) → `int`

Return the number of non-overlapping occurrences of subsection sub in bytes B[start:end]. Optional arguments start and end are interpreted as in slice notation.

decode(*encoding='utf-8'*, *errors='strict'*)

Decode the bytes using the codec registered for encoding.

encoding

The encoding with which to decode the bytes.

errors

The error handling scheme to use for the handling of decoding errors. The default is ‘strict’ meaning that decoding errors raise a `UnicodeDecodeError`. Other possible values are ‘ignore’ and ‘replace’ as well as any other name registered with `codecs.register_error` that can handle `UnicodeDecodeErrors`.

endswith(*suffix*[, *start*[, *end*]]) → `bool`

Return True if B ends with the specified suffix, False otherwise. With optional start, test B beginning at that position. With optional end, stop comparing B at that position. suffix can also be a tuple of bytes to try.

expandtabs(*tabsize=8*)

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in B where subsection sub is found, such that sub is contained within B[start,end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

fromhex()

Create a bytes object from a string of hexadecimal numbers.

Spaces between two numbers are accepted. Example: bytes.fromhex('B9 01EF') -> b'\xb9\x01\xef'.

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

str

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Parameters

member_name (str) – Name to test whether a member exists within the type structure

Return type

bool

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (str) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (List[str]) – List of names to test as to members with those names validity

Return type

bool

hex()

Create a string of hexadecimal numbers from a bytes object.

sep

An optional single character or byte to separate hex bytes.

bytes_per_sep

How many bytes between separators. Positive values count from the right, negative values count from the left.

Example: >>> value = b'\xb9\x01\xef' >>> value.hex() 'b901ef' >>> value.hex(':') 'b9:01:ef' >>> value.hex(':', 2) 'b9:01ef' >>> value.hex(':', -2) 'b901:ef'

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in B where subsection sub is found, such that sub is contained within B[start,end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the subsection is not found.

isalnum() → bool

Return True if all characters in B are alphanumeric and there is at least one character in B, False otherwise.

isalpha() → bool

Return True if all characters in B are alphabetic and there is at least one character in B, False otherwise.

isascii() → bool

Return True if B is empty or all characters in B are ASCII, False otherwise.

isdigit() → bool

Return True if all characters in B are digits and there is at least one character in B, False otherwise.

islower() → bool

Return True if all cased characters in B are lowercase and there is at least one cased character in B, False otherwise.

isspace() → bool

Return True if all characters in B are whitespace and there is at least one character in B, False otherwise.

istitle() → bool

Return True if B is a titlecased string and there is at least one character in B, i.e. uppercase characters may only follow uncased characters and lowercase characters only cased ones. Return False otherwise.

isupper() → bool

Return True if all cased characters in B are uppercase and there is at least one cased character in B, False otherwise.

join(*iterable_of_bytes*, /)

Concatenate any number of bytes objects.

The bytes whose method is called is inserted in between each pair.

The result is returned as a new bytes object.

Example: b'.'.join([b'ab', b'pq', b'rs']) -> b'ab.pq.rs'.

ljust(*width*, *fillchar*=b' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character.

lower() → copy of B

Return a copy of B with all ASCII characters converted to lowercase.

lstrip(*bytes*=None, /)

Strip leading bytes contained in the argument.

If the argument is omitted or None, strip leading ASCII whitespace.

static maketrans(*frm*, *to*, /)

Return a translation table useable for the bytes or bytearray translate method.

The returned table will be one where each byte in frm is mapped to the byte at the same position in to.

The bytes objects frm and to must be of the same length.

partition(*sep*, /)

Partition the bytes into three parts using the given separator.

This will search for the separator *sep* in the bytes. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original bytes object and two empty bytes objects.

removeprefix(*prefix*, /)

Return a bytes object with the given prefix string removed if present.

If the bytes starts with the prefix string, return bytes[len(prefix):]. Otherwise, return a copy of the original bytes.

removesuffix(*suffix*, /)

Return a bytes object with the given suffix string removed if present.

If the bytes ends with the suffix string and that suffix is not empty, return bytes[:-len(suffix)]. Otherwise, return a copy of the original bytes.

replace(*old*, *new*, *count=-1*, /)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *B* where subsection *sub* is found, such that *sub* is contained within *B*[*start*,*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *B* where subsection *sub* is found, such that *sub* is contained within *B*[*start*,*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raise ValueError when the subsection is not found.

rjust(*width*, *fillchar=b' '*, /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character.

rpartition(*sep*, /)

Partition the bytes into three parts using the given separator.

This will search for the separator *sep* in the bytes, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty bytes objects and the original bytes object.

rsplit(*sep=None*, *maxsplit=-1*)

Return a list of the sections in the bytes, using *sep* as the delimiter.

sep

The delimiter according which to split the bytes. None (the default value) means split on ASCII whitespace characters (space, tab, return, newline, formfeed, vertical tab).

maxsplit

Maximum number of splits to do. -1 (the default value) means no limit.

Splitting is done starting at the end of the bytes and working to the front.

rstrip(*bytes=None, /*)

Strip trailing bytes contained in the argument.

If the argument is omitted or None, strip trailing ASCII whitespace.

split(*sep=None, maxsplit=-1*)

Return a list of the sections in the bytes, using sep as the delimiter.

sep

The delimiter according which to split the bytes. None (the default value) means split on ASCII whitespace characters (space, tab, return, newline, formfeed, vertical tab).

maxsplit

Maximum number of splits to do. -1 (the default value) means no limit.

splitlines(*keepends=False*)

Return a list of the lines in the bytes, breaking at line boundaries.

Line breaks are not included in the resulting list unless keepends is given and true.

startswith(*prefix[, start[, end]]*) → bool

Return True if B starts with the specified prefix, False otherwise. With optional start, test B beginning at that position. With optional end, stop comparing B at that position. prefix can also be a tuple of bytes to try.

strip(*bytes=None, /*)

Strip leading and trailing bytes contained in the argument.

If the argument is omitted or None, strip leading and trailing ASCII whitespace.

swapcase() → copy of B

Return a copy of B with uppercase ASCII characters converted to lowercase ASCII and vice versa.

title() → copy of B

Return a titlecased version of B, i.e. ASCII words start with uppercase characters, all remaining cased characters have lowercase.

translate(*table, /, delete=b''*)

Return a copy with each character mapped by the given translation table.

table

Translation table, which must be a bytes object of length 256.

All characters occurring in the optional argument delete are removed. The remaining characters are mapped through the given translation table.

upper() → copy of B

Return a copy of B with all ASCII characters converted to uppercase.

property vol: [*ReadOnlyMapping*](#)

Returns the volatility specific object information.

write(*value*)

Writes the object into the layer of the context at the current offset.

Return type

[*ObjectInterface*](#)

zfill(width, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The original string is never truncated.

class Char(context, type_name, object_info, data_format)

Bases: [PrimitiveObject](#), [int](#)

Primitive Object that handles characters.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** ([ContextInterface](#)) – The context associated with the object
- **type_name** ([str](#)) – The name of the type structure for the object
- **object_info** ([ObjectInformation](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: [VolTemplateProxy](#)

abstract classmethod child_template(template, child)

Returns the template of the child member from the parent.

Return type

[Template](#)

abstract classmethod children(template)

Returns the children of the template.

Return type

[List\[Template\]](#)

abstract classmethod has_member(template, member_name)

Returns whether the object would contain a member called member_name.

Return type

[bool](#)

abstract classmethod relative_child_offset(template, child)

Returns the relative offset from the head of the parent data to the child member.

Return type

[int](#)

abstract classmethod replace_child(template, old_child, new_child)

Substitutes the old_child for the new_child.

Return type

[None](#)

classmethod size(template)

Returns the size of the templated object.

Return type

[int](#)

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

cast(new_type_name, **additional)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Parameters

member_name (`str`) – Name to test whether a member exists within the type structure

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

imag

the imaginary part of a complex number

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(length=1, byteorder='big', *, signed=False)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is ‘big’, the most significant byte is at the beginning of the byte array. If byteorder is ‘little’, the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use ‘big’.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the object into the layer of the context at the current offset.

Return type

ObjectInterface

class ClassType(context, type_name, object_info, size, members)

Bases: *AggregateType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(template, child)

Returns the template of a child to its parent.

Return type

Template

classmethod children(template)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(template, member_name)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(template, child)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(template, old_child, new_child)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(template)

Method to return the size of this type.

Return type

int

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

str

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class DataFormatInfo(*length, byteorder, signed*)

Bases: *tuple*

Create new instance of DataFormatInfo(length, byteorder, signed)

byteorder

Alias for field number 1

count(*value*, /)

Return number of occurrences of value.

index(*value*, *start*=0, *stop*=9223372036854775807, /)

Return first index of value.

Raises ValueError if the value is not present.

length

Alias for field number 0

signed

Alias for field number 2

class Enumeration(*context*, *type_name*, *object_info*, *base_type*, *choices*)

Bases: [ObjectInterface](#), [int](#)

Returns an object made up of choices.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** ([ContextInterface](#)) – The context associated with the object
- **type_name** ([str](#)) – The name of the type structure for the object
- **object_info** ([ObjectInformation](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: [VolTemplateProxy](#)

abstract classmethod child_template(*template*, *child*)

Returns the template of the child member from the parent.

Return type

[Template](#)

classmethod children(*template*)

Returns the children of the template.

Return type

[List](#)[[Template](#)]

abstract classmethod has_member(*template*, *member_name*)

Returns whether the object would contain a member called member_name.

Return type

[bool](#)

classmethod lookup(*template*, *value*)

Looks up an individual value and returns the associated name.

If multiple identifiers map to the same value, the first matching identifier will be returned

Return type

[str](#)

abstract classmethod relative_child_offset(*template, child*)

Returns the relative offset from the head of the parent data to the child member.

Return type

`int`

classmethod replace_child(*template, old_child, new_child*)

Substitutes the *old_child* for the *new_child*.

Return type

`None`

classmethod size(*template*)

Returns the size of the template object.

Return type

`int`

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

ObjectInterface

Note: If new type name does not include a symbol table, the symbol table for the current object is used

property choices: `Dict[str, int]`

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

property description: `str`

Returns the chosen name for the value this object contains.

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Parameters

member_name (`str`) – Name to test whether a member exists within the type structure

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**imag**

the imaginary part of a complex number

property is_valid_choice: `bool`

Returns whether the value for the object is a valid choice

lookup(*value=None*)

Looks up an individual value and returns the associated name.

If multiple identifiers map to the same value, the first matching identifier will be returned

Return type`str`**numerator**

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If `byteorder` is 'big', the most significant byte is at the beginning of the byte array. If `byteorder` is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If `signed` is `False` and a negative integer is given, an `OverflowError` is raised.

property vol: `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class Float(*context, type_name, object_info, data_format*)

Bases: `PrimitiveObject`, `float`

Primitive Object that handles double or floating point numbers.

Constructs an Object adhering to the `ObjectInterface`.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

abstract classmethod child_template(*template, child*)

Returns the template of the child member from the parent.

Return type

Template

abstract classmethod children(*template*)

Returns the children of the template.

Return type

List[Template]

abstract classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

abstract classmethod relative_child_offset(*template, child*)

Returns the relative offset from the head of the parent data to the child member.

Return type

int

abstract classmethod replace_child(*template, old_child, new_child*)

Substitutes the old_child for the new_child.

Return type

None

classmethod size(*template*)

Returns the size of the templated object.

Return type

int

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original float and with a positive denominator.

Raise OverflowError on infinities and a ValueError on NaNs.

```
>>> (10.0).as_integer_ratio()
(10, 1)
>>> (0.0).as_integer_ratio()
(0, 1)
>>> (-.25).as_integer_ratio()
(-1, 4)
```

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

conjugate()

Return self, the complex conjugate of any float.

fromhex()

Create a floating-point number from a hexadecimal string.

```
>>> float.fromhex('0x1.ffffp10')
2047.984375
>>> float.fromhex('-0x1p-1074')
-5e-324
```

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

str

has_member(member_name)

Returns whether the object would contain a member called member_name.

Parameters

member_name (str) – Name to test whether a member exists within the type structure

Return type

bool

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (str) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (List[str]) – List of names to test as to members with those names validity

Return type

bool

hex()

Return a hexadecimal representation of a floating-point number.

```
>>> (-0.1).hex()
'-0x1.999999999999ap-4'
>>> 3.14159.hex()
'0x1.921f9f01b866ep+1'
```

imag

the imaginary part of a complex number

is_integer()

Return True if the float is an integer.

real

the real part of a complex number

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the object into the layer of the context at the current offset.

Return type

ObjectInterface

class Function(context, type_name, object_info, **kwargs)

Bases: *ObjectInterface*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *object*

A container for proxied methods that the ObjectTemplate of this object will call. This is primarily to keep methods together for easy organization/management, there is no significant need for it to be a separate class.

The methods of this class *must* be class methods rather than standard methods, to allow for code reuse. Each method also takes a template since the templates may contain the necessary data about the yet-to-be-constructed object. It allows objects to control how their templates respond without needing to write new templates for each and every potential object type.

abstract classmethod child_template(template, child)

Returns the template of the child member from the parent.

Return type

Template

abstract classmethod children(template)

Returns the children of the template.

Return type

List[Template]

abstract classmethod has_member(template, member_name)

Returns whether the object would contain a member called member_name.

Return type

bool

abstract classmethod relative_child_offset(template, child)

Returns the relative offset from the head of the parent data to the child member.

Return type

int

abstract classmethod `replace_child(template, old_child, new_child)`

Substitutes the old_child for the new_child.

Return type

`None`

abstract classmethod `size(template)`

Returns the size of the template object.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Parameters

member_name (`str`) – Name to test whether a member exists within the type structure

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

property `vol: ReadOnlyMapping`

Returns the volatility specific object information.

abstract write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class Integer(*context, type_name, object_info, data_format*)

Bases: *PrimitiveObject*, *int*

Primitive Object that handles standard numeric types.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

abstract classmethod child_template(*template, child*)

Returns the template of the child member from the parent.

Return type

Template

abstract classmethod children(*template*)

Returns the children of the template.

Return type

List[Template]

abstract classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

abstract classmethod relative_child_offset(*template, child*)

Returns the relative offset from the head of the parent data to the child member.

Return type

int

abstract classmethod replace_child(*template, old_child, new_child*)

Substitutes the old_child for the new_child.

Return type

None

classmethod size(*template*)

Returns the size of the templated object.

Return type

int

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

cast(new_type_name, **additional)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Parameters

member_name (`str`) – Name to test whether a member exists within the type structure

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

imag

the imaginary part of a complex number

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(length=1, byteorder='big', *, signed=False)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is ‘big’, the most significant byte is at the beginning of the byte array. If byteorder is ‘little’, the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use ‘big’.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the object into the layer of the context at the current offset.

Return type

ObjectInterface

class Pointer(context, type_name, object_info, data_format, subtype=None)

Bases: *Integer*

Pointer which points to another object.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

abstract classmethod child_template(template, child)

Returns the template of the child member from the parent.

Return type

Template

classmethod children(template)

Returns the children of the template.

Return type

List[Template]

classmethod has_member(template, member_name)

Returns whether the object would contain a member called member_name.

Return type

bool

abstract classmethod relative_child_offset(template, child)

Returns the relative offset from the head of the parent data to the child member.

Return type

int

classmethod replace_child(template, old_child, new_child)

Substitutes the old_child for the new_child.

Return type

None

classmethod size(template)

Returns the size of the template object.

Return type`int`**as_integer_ratio()**

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

cast(new_type_name, **additional)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

dereference(layer_name=None)

Dereferences the pointer.

Layer_name identifies the appropriate layer within the context that the pointer points to. If layer_name is None, it defaults to the same layer that the pointer is currently instantiated in.

Return type*ObjectInterface*

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(*member_name*)

Returns whether the dereferenced type has this member.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

imag

the imaginary part of a complex number

is_readable(*layer_name=None*)

Determines whether the address of this pointer can be read from memory.

Return type

`bool`

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If `byteorder` is 'big', the most significant byte is at the beginning of the byte array. If `byteorder` is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If `signed` is `False` and a negative integer is given, an `OverflowError` is raised.

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the object into the layer of the context at the current offset.

Return type

ObjectInterface

class PrimitiveObject(*context, type_name, object_info, data_format*)

Bases: *ObjectInterface*

PrimitiveObject is an interface for any objects that should simulate a Python primitive.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

abstract classmethod child_template(*template, child*)

Returns the template of the child member from the parent.

Return type

Template

abstract classmethod children(*template*)

Returns the children of the template.

Return type

List[Template]

abstract classmethod has_member(*template*, *member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

abstract classmethod relative_child_offset(*template*, *child*)

Returns the relative offset from the head of the parent data to the child member.

Return type

`int`

abstract classmethod replace_child(*template*, *old_child*, *new_child*)

Substitutes the *old_child* for the *new_child*.

Return type

`None`

classmethod size(*template*)

Returns the size of the templated object.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the *table_name* is not valid within the object’s context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Parameters

member_name (`str`) – Name to test whether a member exists within the type structure

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the object into the layer of the context at the current offset.

Return type

ObjectInterface

class String(*context, type_name, object_info, max_length=1, encoding='utf-8', errors='strict'*)

Bases: *PrimitiveObject, str*

Primitive Object that handles string values.

Parameters

max_length (*int*) – specifies the maximum possible length that the string could hold within memory (for multibyte characters, this will not be the maximum length of the string)

Constructs an Object adhering to the *ObjectInterface*.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

abstract classmethod child_template(*template, child*)

Returns the template of the child member from the parent.

Return type

Template

abstract classmethod children(*template*)

Returns the children of the template.

Return type

List[Template]

abstract classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

bool

abstract classmethod relative_child_offset(*template, child*)

Returns the relative offset from the head of the parent data to the child member.

Return type

int

abstractmethod `replace_child(template, old_child, new_child)`

Substitutes the old_child for the new_child.

Return type

`None`

classmethod `size(template)`

Returns the size of the templated object.

Return type

`int`

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

center(*width*, *fillchar*=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → `int`

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode(*encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(*suffix*[, *start*[, *end*]]) → `bool`

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → `int`

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format(*args, **kwargs) → str

Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

format_map(mapping) → str

Return a formatted version of S, using substitutions from mapping. The substitutions are identified by braces ('{' and '}').

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

str

has_member(member_name)

Returns whether the object would contain a member called member_name.

Parameters

member_name (str) – Name to test whether a member exists within the type structure

Return type

bool

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (str) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (List[str]) – List of names to test as to members with those names validity

Return type

bool

index(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call `keyword.iskeyword(s)` to test whether string `s` is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in `repr()` or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(*iterable*, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(*width*, *fillchar*=' ', /)

Return a left-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(*chars*=None, /)

Return a copy of the string with leading whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in *x* will be mapped to the character at the same position in *y*. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(*prefix*, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count*=-1, /)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises *ValueError* when the substring is not found.

rjust(*width*, *fillchar*=' ', /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including *n* *r* *t* *f* and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars*=None, /)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

split(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including *n* *r* *t* *f* and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, *str.split()* is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends*=False)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

strip(*chars*=None, /)

Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table*, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the object into the layer of the context at the current offset.

Return type

ObjectInterface

zfill(*width*, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class **StructType**(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: *AggregateType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (str) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxyBases: *VolTemplateProxy***classmethod child_template**(*template, child*)

Returns the template of a child to its parent.

Return type*Template***classmethod children**(*template*)

Method to list children of a template.

Return type*List[Template]***classmethod has_member**(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type*bool***classmethod relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type*int***classmethod replace_child**(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type*None***classmethod size**(*template*)

Method to return the size of this type.

Return type*int***cast**(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

ObjectInterface

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type*str***has_member**(*member_name*)

Returns whether the object would contain a member called member_name.

Return type*bool*

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class UnionType(*context, type_name, object_info, size, members*)

Bases: *AggregateType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called *member_name*.

Return type`bool`**classmethod** `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type`int`**classmethod** `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type`None`**classmethod** `size(template)`

Method to return the size of this type.

Return type`int`**cast**(*new_type_name*, ***additional*)Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type`str`**has_member**(*member_name*)

Returns whether the object would contain a member called member_name.

Return type`bool`**has_valid_member**(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters**member_name** (`str`) – Name of the member to test access to determine if the member is valid or not**Return type**`bool`**has_valid_members**(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters**member_names** (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**member**(*attr*='member')

Specifically named method for retrieving members.

Return type`object`**property vol:** `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class Void(*context*, *type_name*, *object_info*, ***kwargs*)

Bases: `ObjectInterface`

Returns an object to represent void/unknown types.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: `VolTemplateProxy`

abstract classmethod child_template(*template*, *child*)

Returns the template of the child member from the parent.

Return type`Template`**abstract classmethod children**(*template*)

Returns the children of the template.

Return type`List[Template]`**abstract classmethod has_member**(*template*, *member_name*)

Returns whether the object would contain a member called member_name.

Return type`bool`**abstract classmethod relative_child_offset**(*template*, *child*)

Returns the relative offset from the head of the parent data to the child member.

Return type`int`**abstract classmethod replace_child**(*template*, *old_child*, *new_child*)

Substitutes the old_child for the new_child.

Return type`None`

classmethod `size(template)`

Dummy size for Void objects.

According to <http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf>, void is an incomplete type, and therefore `sizeof(void)` should fail. However, we need to be able to construct voids to be able to cast them, so we return a useless size. It shouldn't cause errors, but it also shouldn't be common, it is logged at the lowest level.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Parameters

member_name (`str`) – Name to test whether a member exists within the type structure

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

property `vol: ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Dummy method that does nothing for Void objects.

Return type

`None`

convert_data_to_value(*data*, *struct_type*, *data_format*)

Converts a series of bytes to a particular type of value.

Return type

`Union[int, float, bytes, str, bool]`

convert_value_to_data(*value*, *struct_type*, *data_format*)

Converts a particular value to a series of bytes.

Return type

`bytes`

Submodules

volatility3.framework.objects.templates module

class `ObjectTemplate`(*object_class*, *type_name*, ***arguments*)

Bases: `Template`

Factory class that produces objects that adhere to the Object interface on demand.

This is effectively a method of currying, but adds more structure to avoid abuse. It also allows inspection of information that should already be known:

- Type size
- Members
- etc

Stores the keyword arguments for later object creation.

child_template(*child*)

Returns the template of a child of the templated object (see `VolTemplateProxy`)

Return type

`Template`

property children: `List[Template]`

`~volatility3.framework.interfaces.objects.ObjectInterface.VolTemplateProxy)`

Type

Returns the children of the templated object (see

Type

`class`

clone()

Returns a copy of the original Template as constructed (without *update_vol* additions having been made)

Return type

`Template`

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

relative_child_offset(*child*)

Returns the relative offset of a child of the templated object (see `VolTemplateProxy`)

Return type

`int`

replace_child(*old_child*, *new_child*)

Replaces *old_child* for *new_child* in the templated object's child list (see `VolTemplateProxy`)

Return type

`None`

property size: `int`

`~volatility.framework.interfaces.objects.ObjectInterface.VolTemplateProxy`

Type

Returns the children of the templated object (see

Type

`class`

update_vol(***new_arguments*)

Updates the keyword arguments with values that will **not** be carried across to clones.

Return type

`None`

property vol: `ReadOnlyMapping`

Returns a volatility information object, much like the `ObjectInformation` provides.

class ReferenceTemplate(*type_name*, ***arguments*)

Bases: `Template`

Factory class that produces objects based on a delayed reference type.

Attempts to access any standard attributes of a resolved template will result in a `SymbolError`.

Stores the keyword arguments for later object creation.

child_template(**args*, ***kwargs*)

Referenced symbols must be appropriately resolved before they can provide information such as size This is because the size request has no context within which to determine the actual symbol structure.

Return type

`Any`

property children: `List[Template]`

The children of this template (such as member types, sub-types and base-types where they are relevant).

Used to traverse the template tree.

clone()

Returns a copy of the original Template as constructed (without *update_vol* additions having been made)

Return type

`Template`

has_member(*args, **kwargs)

Referenced symbols must be appropriately resolved before they can provide information such as size This is because the size request has no context within which to determine the actual symbol structure.

Return type

Any

relative_child_offset(*args, **kwargs)

Referenced symbols must be appropriately resolved before they can provide information such as size This is because the size request has no context within which to determine the actual symbol structure.

Return type

Any

replace_child(*args, **kwargs)

Referenced symbols must be appropriately resolved before they can provide information such as size This is because the size request has no context within which to determine the actual symbol structure.

Return type

Any

property size: *Any*

Referenced symbols must be appropriately resolved before they can provide information such as size This is because the size request has no context within which to determine the actual symbol structure.

update_vol(**new_arguments)

Updates the keyword arguments with values that will **not** be carried across to clones.

Return type

None

property vol: *ReadOnlyMapping*

Returns a volatility information object, much like the *ObjectInformation* provides.

volatility3.framework.objects.utility module

array_of_pointers(array, count, subtype, context)

Takes an object, and recasts it as an array of pointers to subtype.

Return type

ObjectInterface

array_to_string(array, count=None, errors='replace')

Takes a volatility Array of characters and returns a string.

Return type

ObjectInterface

pointer_to_string(pointer, count, errors='replace')

Takes a volatility Pointer to characters and returns a string.

volatility3.framework.plugins package

All core generic plugins.

These modules should only be imported from volatility3.plugins NOT volatility3.framework.plugins

construct_plugin(*context, automagics, plugin, base_config_path, progress_callback, open_method*)

Constructs a plugin object based on the parameters.

Clever magic figures out how to fulfill each requirement that might not be fulfilled

Parameters

- **context** (*ContextInterface*) – The volatility context to operate on
- **automagics** (*List[AutomagicInterface]*) – A list of automagic modules to run to augment the context
- **plugin** (*Type[PluginInterface]*) – The plugin to run
- **base_config_path** (*str*) – The path within the context’s config containing the plugin’s configuration
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – Callback function to provide feedback for ongoing processes
- **open_method** (*Type[FileHandlerInterface]*) – class to provide context manager for opening the file

Return type

PluginInterface

Returns

The constructed plugin object

Subpackages

Submodules

volatility3.framework.renderers package

Renderers.

Renderers display the unified output format in some manner (be it text or file or graphical output

class ColumnSortKey(*treegrid, column_name, ascending=True*)

Bases: *ColumnSortKey*

ascending: *bool* = *True*

class NotApplicableValue

Bases: *BaseAbsentValue*

Class that represents values which are empty because they don’t make sense for this node.

class NotAvailableValue

Bases: *BaseAbsentValue*

Class that represents values which cannot be provided now (but might in a future run)

This might occur when information packed with volatility (such as symbol information) is not available, but a future version or a different run may later have that information available (ie, it could be applicable, but we can't get it and it's not because it's unreadable or unparsable). Unreadable and Unparsable should be used in preference, and only if neither fits should this be used.

RowStructureConstructor(*names*)

class TreeGrid(*columns*, *generator*)

Bases: [*TreeGrid*](#)

Class providing the interface for a TreeGrid (which contains TreeNodes)

The structure of a TreeGrid is designed to maintain the structure of the tree in a single object. For this reason each TreeNode does not hold its children, they are managed by the top level object. This leaves the Nodes as simple data carries and prevents them being used to manipulate the tree as a whole. This is a data structure, and is not expected to be modified much once created.

Carrying the children under the parent makes recursion easier, but then every node is its own little tree and must have all the supporting tree functions. It also allows for a node to be present in several different trees, and to create cycles.

Constructs a TreeGrid object using a specific set of columns.

The TreeGrid itself is a root element, that can have children but no values. The TreeGrid does *not* contain any information about formatting, these are up to the renderers and plugins.

Parameters

- **columns** (`List[Tuple[str, Union[Type[int], Type[str], Type[float], Type[bytes], Type[datetime], Type[BaseAbsentValue], Type[Disassembly]]]]`) – A list of column tuples made up of (name, type).
- **generator** (`Optional[Iterable[Tuple[int, Tuple]]]`) – An iterable containing row for a tree grid, each row contains a indent level followed by the values for each column in order.

base_types: `ClassVar[Tuple] = (<class 'int'>, <class 'str'>, <class 'float'>, <class 'bytes'>, <class 'datetime.datetime'>, <class 'volatility3.framework.interfaces.renderers.Disassembly'>)`

children(*node*)

Returns the subnodes of a particular node in order.

Return type

`List[TreeNode]`

property columns: `List[Column]`

Returns the available columns and their ordering and types.

is_ancestor(*node*, *descendant*)

Returns true if descendent is a child, grandchild, etc of node.

max_depth()

Returns the maximum depth of the tree.

static path_depth(*node*)

Returns the path depth of a particular node.

Return type

`int`

path_sep = '|'

populate(*function=None, initial_accumulator=None, fail_on_errors=True*)

Populates the tree by consuming the TreeGrid's construction generator. *Func* is called on every node, so can be used to create output on demand.

This is equivalent to a one-time visit.

Parameters

- **function** (*Callable[[TreeNode, TypeVar(_Type)], TypeVar(_Type)]*) – The visitor to be called on each row of the treegrid
- **initial_accumulator** (*Any*) – The initial value for an accumulator passed to the visitor to allow it to maintain state
- **fail_on_errors** (*bool*) – A boolean defining whether exceptions should be caught or bubble up

Return type

Optional[Exception]

property populated: *bool*

Indicates that population has completed and the tree may now be manipulated separately.

property row_count: *int*

Returns the number of rows populated.

static sanitize_name(*text*)

Method used to sanitize column names for TreeNodes.

Return type

str

values(*node*)

Returns the values for a particular node.

The values returned are mutable,

visit(*node, function, initial_accumulator, sort_key=None*)

Visits all the nodes in a tree, calling *function* on each one.

function should have the signature *function*(*node, accumulator*) and return *new_accumulator*. If accumulators are not needed, the function must still accept a second parameter.

The order of that the nodes are visited is always depth first, however, the order children are traversed can be set based on a *sort_key* function which should accept a node's values and return something that can be sorted to receive the desired order (similar to the *sort/sorted* key).

We use the private *_find_children* function so that we don't have to re-traverse the tree for every node we descend further down

class TreeNode(*path, treegrid, parent, values*)

Bases: *TreeNode*

Class representing a particular node in a tree grid.

Initializes the *TreeNode*.

asdict()

Returns the contents of the node as a dictionary

Return type

Dict[str, Any]

count(*value*) → integer -- return number of occurrences of value

index(*value*[, *start*[, *stop*]]) → integer -- return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

property parent: `TreeNode` | `None`

Returns the parent node of this node or None.

property path: `str`

Returns a path identifying string.

This should be seen as opaque by external classes, Parsing of path locations based on this string are not guaranteed to remain stable.

path_changed(*path*, *added=False*)

Updates the path based on the addition or removal of a node higher up in the tree.

This should only be called by the containing TreeGrid and expects to only be called for affected nodes.

Return type

`None`

property path_depth: `int`

Return the path depth of the current node.

property values: `List[Type[int] | Type[str] | Type[float] | Type[bytes] | Type[datetime] | Type[BaseAbsentValue] | Type[Disassembly]]`

Returns the list of values from the particular node, based on column index.

class UnparsableValue

Bases: `BaseAbsentValue`

Class that represents values which are empty because the data cannot be interpreted correctly.

class UnreadableValue

Bases: `BaseAbsentValue`

Class that represents values which are empty because the data cannot be read.

Submodules

volatility3.framework.renderers.conversion module

convert_ipv4(*ip_as_integer*)

convert_ipv6(*packed_ip*)

convert_network_four_tuple(*family*, *four_tuple*)

Converts the connection four_tuple: (source ip, source port, dest ip, dest port)

into their string equivalents. IP addresses are expected as a tuple of unsigned shorts Ports are converted to proper endianness as well

convert_port(*port_as_integer*)

round(*addr*, *align*, *up=False*)

Round an address up or down based on an alignment.

Parameters

- **addr** (`int`) – the address
- **align** (`int`) – the alignment value
- **up** (`bool`) – Whether to round up or not

Return type

`int`

Returns

The aligned address

unixtime_to_datetime(*unixtime*)

Return type

`Union[BaseAbsentValue, datetime]`

wintime_to_datetime(*wintime*)

Return type

`Union[BaseAbsentValue, datetime]`

volatility3.framework.renderers.format_hints module

The official list of format hints that text renderers and plugins can rely upon existing within the framework.

These hints allow a plugin to indicate how they would like data from a particular column to be represented.

Text renderers should attempt to honour all hints provided in this module where possible

class Bin

Bases: `int`

A class to indicate that the integer value should be represented as a binary value.

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

class Hex

Bases: `int`

A class to indicate that the integer value should be represented as a hexadecimal value.

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

class HexBytes

Bases: `bytes`

A class to indicate that the bytes should be display in an extended format showing hexadecimal and ascii printable display.

capitalize() → copy of B

Return a copy of B with only its first character capitalized (ASCII) and the rest lower-cased.

center(*width, fillchar=b' ', /*)

Return a centered string of length width.

Padding is done using the specified fill character.

count(*sub[, start[, end]]*) → int

Return the number of non-overlapping occurrences of subsection sub in bytes B[start:end]. Optional arguments start and end are interpreted as in slice notation.

decode(*encoding='utf-8', errors='strict'*)

Decode the bytes using the codec registered for encoding.

encoding

The encoding with which to decode the bytes.

errors

The error handling scheme to use for the handling of decoding errors. The default is 'strict' meaning that decoding errors raise a `UnicodeDecodeError`. Other possible values are 'ignore' and 'replace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeDecodeErrors`.

endswith(*suffix*[, *start*[, *end*]]) → bool

Return True if B ends with the specified suffix, False otherwise. With optional start, test B beginning at that position. With optional end, stop comparing B at that position. *suffix* can also be a tuple of bytes to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in B where subsection *sub* is found, such that *sub* is contained within B[start,end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

fromhex()

Create a bytes object from a string of hexadecimal numbers.

Spaces between two numbers are accepted. Example: `bytes.fromhex('B9 01EF') -> b'\xb9\x01\xef'`.

hex()

Create a string of hexadecimal numbers from a bytes object.

sep

An optional single character or byte to separate hex bytes.

bytes_per_sep

How many bytes between separators. Positive values count from the right, negative values count from the left.

Example: `>>> value = b'\xb9\x01\xef' >>> value.hex() 'b901ef' >>> value.hex(':') 'b9:01:ef' >>> value.hex(':', 2) 'b9:01ef' >>> value.hex(':', -2) 'b901:ef'`

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in B where subsection *sub* is found, such that *sub* is contained within B[start,end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the subsection is not found.

isalnum() → bool

Return True if all characters in B are alphanumeric and there is at least one character in B, False otherwise.

isalpha() → bool

Return True if all characters in B are alphabetic and there is at least one character in B, False otherwise.

isascii() → bool

Return True if B is empty or all characters in B are ASCII, False otherwise.

isdigit() → bool

Return True if all characters in B are digits and there is at least one character in B, False otherwise.

islower() → bool

Return True if all cased characters in B are lowercase and there is at least one cased character in B, False otherwise.

isspace() → bool

Return True if all characters in B are whitespace and there is at least one character in B, False otherwise.

istitle() → bool

Return True if B is a titlecased string and there is at least one character in B, i.e. uppercase characters may only follow uncased characters and lowercase characters only cased ones. Return False otherwise.

isupper() → bool

Return True if all cased characters in B are uppercase and there is at least one cased character in B, False otherwise.

join(iterable_of_bytes, /)

Concatenate any number of bytes objects.

The bytes whose method is called is inserted in between each pair.

The result is returned as a new bytes object.

Example: `b'.'.join([b'ab', b'pq', b'rs']) -> b'ab.pq.rs'`.

ljust(width, fillchar=b' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character.

lower() → copy of B

Return a copy of B with all ASCII characters converted to lowercase.

lstrip(bytes=None, /)

Strip leading bytes contained in the argument.

If the argument is omitted or None, strip leading ASCII whitespace.

static maketrans(frm, to, /)

Return a translation table useable for the bytes or bytearray translate method.

The returned table will be one where each byte in frm is mapped to the byte at the same position in to.

The bytes objects frm and to must be of the same length.

partition(sep, /)

Partition the bytes into three parts using the given separator.

This will search for the separator sep in the bytes. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original bytes object and two empty bytes objects.

removeprefix(prefix, /)

Return a bytes object with the given prefix string removed if present.

If the bytes starts with the prefix string, return `bytes[len(prefix):]`. Otherwise, return a copy of the original bytes.

removesuffix(suffix, /)

Return a bytes object with the given suffix string removed if present.

If the bytes ends with the suffix string and that suffix is not empty, return `bytes[:-len(suffix)]`. Otherwise, return a copy of the original bytes.

replace(old, new, count=-1, /)

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in B where subsection sub is found, such that sub is contained within B[start,end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in B where subsection sub is found, such that sub is contained within B[start,end]. Optional arguments start and end are interpreted as in slice notation.

Raise ValueError when the subsection is not found.

rjust(*width*, *fillchar*=b' ', /)

Return a right-justified string of length width.

Padding is done using the specified fill character.

rpartition(*sep*, /)

Partition the bytes into three parts using the given separator.

This will search for the separator sep in the bytes, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty bytes objects and the original bytes object.

rsplit(*sep*=None, *maxsplit*=-1)

Return a list of the sections in the bytes, using sep as the delimiter.

sep

The delimiter according which to split the bytes. None (the default value) means split on ASCII whitespace characters (space, tab, return, newline, formfeed, vertical tab).

maxsplit

Maximum number of splits to do. -1 (the default value) means no limit.

Splitting is done starting at the end of the bytes and working to the front.

rstrip(*bytes*=None, /)

Strip trailing bytes contained in the argument.

If the argument is omitted or None, strip trailing ASCII whitespace.

split(*sep*=None, *maxsplit*=-1)

Return a list of the sections in the bytes, using sep as the delimiter.

sep

The delimiter according which to split the bytes. None (the default value) means split on ASCII whitespace characters (space, tab, return, newline, formfeed, vertical tab).

maxsplit

Maximum number of splits to do. -1 (the default value) means no limit.

splitlines(*keepends*=False)

Return a list of the lines in the bytes, breaking at line boundaries.

Line breaks are not included in the resulting list unless keepends is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if B starts with the specified prefix, False otherwise. With optional start, test B beginning at that position. With optional end, stop comparing B at that position. *prefix* can also be a tuple of bytes to try.

strip(*bytes*=None, /)

Strip leading and trailing bytes contained in the argument.

If the argument is omitted or None, strip leading and trailing ASCII whitespace.

swapcase() → copy of B

Return a copy of B with uppercase ASCII characters converted to lowercase ASCII and vice versa.

title() → copy of B

Return a titlecased version of B, i.e. ASCII words start with uppercase characters, all remaining cased characters have lowercase.

translate(*table*, /, *delete*=b'')

Return a copy with each character mapped by the given translation table.

table

Translation table, which must be a bytes object of length 256.

All characters occurring in the optional argument *delete* are removed. The remaining characters are mapped through the given translation table.

upper() → copy of B

Return a copy of B with all ASCII characters converted to uppercase.

zfill(*width*, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The original string is never truncated.

class MultiTypeData(*original*, *encoding*='utf-16-le', *split_nulls*=False, *show_hex*=False)

Bases: `bytes`

The contents are supposed to be a string, but may contain binary data.

capitalize() → copy of B

Return a copy of B with only its first character capitalized (ASCII) and the rest lower-cased.

center(*width*, *fillchar*=b' ', /)

Return a centered string of length *width*.

Padding is done using the specified fill character.

count(*sub*[, *start*[, *end*]]) → int

Return the number of non-overlapping occurrences of subsection *sub* in bytes B[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation.

decode(*encoding*='utf-8', *errors*='strict')

Decode the bytes using the codec registered for encoding.

encoding

The encoding with which to decode the bytes.

errors

The error handling scheme to use for the handling of decoding errors. The default is 'strict' meaning that decoding errors raise a `UnicodeDecodeError`. Other possible values are 'ignore' and 'replace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeDecodeErrors`.

endswith(*suffix*[, *start*[, *end*]]) → bool

Return True if B ends with the specified suffix, False otherwise. With optional start, test B beginning at that position. With optional end, stop comparing B at that position. *suffix* can also be a tuple of bytes to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in B where subsection *sub* is found, such that *sub* is contained within B[start,end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

fromhex()

Create a bytes object from a string of hexadecimal numbers.

Spaces between two numbers are accepted. Example: `bytes.fromhex('B9 01EF') -> b'\xb9\x01\xef'`.

hex()

Create a string of hexadecimal numbers from a bytes object.

sep

An optional single character or byte to separate hex bytes.

bytes_per_sep

How many bytes between separators. Positive values count from the right, negative values count from the left.

Example: `>>> value = b'\xb9\x01\xef' >>> value.hex() 'b901ef' >>> value.hex(':') 'b9:01:ef' >>> value.hex(':', 2) 'b9:01ef' >>> value.hex(':', -2) 'b901:ef'`

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in B where subsection *sub* is found, such that *sub* is contained within B[start,end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the subsection is not found.

isalnum() → bool

Return True if all characters in B are alphanumeric and there is at least one character in B, False otherwise.

isalpha() → bool

Return True if all characters in B are alphabetic and there is at least one character in B, False otherwise.

isascii() → bool

Return True if B is empty or all characters in B are ASCII, False otherwise.

isdigit() → bool

Return True if all characters in B are digits and there is at least one character in B, False otherwise.

islower() → bool

Return True if all cased characters in B are lowercase and there is at least one cased character in B, False otherwise.

isspace() → bool

Return True if all characters in B are whitespace and there is at least one character in B, False otherwise.

istitle() → bool

Return True if B is a titlecased string and there is at least one character in B, i.e. uppercase characters may only follow uncased characters and lowercase characters only cased ones. Return False otherwise.

isupper() → bool

Return True if all cased characters in B are uppercase and there is at least one cased character in B, False otherwise.

join(*iterable_of_bytes*, /)

Concatenate any number of bytes objects.

The bytes whose method is called is inserted in between each pair.

The result is returned as a new bytes object.

Example: `b''.join([b'ab', b'pq', b'rs']) -> b'ab.pq.rs'`.

ljust(*width*, *fillchar=b' '*, /)

Return a left-justified string of length width.

Padding is done using the specified fill character.

lower() → copy of B

Return a copy of B with all ASCII characters converted to lowercase.

lstrip(*bytes=None*, /)

Strip leading bytes contained in the argument.

If the argument is omitted or None, strip leading ASCII whitespace.

static maketrans(*frm*, *to*, /)

Return a translation table useable for the bytes or bytearray translate method.

The returned table will be one where each byte in frm is mapped to the byte at the same position in to.

The bytes objects frm and to must be of the same length.

partition(*sep*, /)

Partition the bytes into three parts using the given separator.

This will search for the separator sep in the bytes. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original bytes object and two empty bytes objects.

removeprefix(*prefix*, /)

Return a bytes object with the given prefix string removed if present.

If the bytes starts with the prefix string, return bytes[len(prefix):]. Otherwise, return a copy of the original bytes.

removesuffix(*suffix*, /)

Return a bytes object with the given suffix string removed if present.

If the bytes ends with the suffix string and that suffix is not empty, return bytes[:-len(suffix)]. Otherwise, return a copy of the original bytes.

replace(*old*, *new*, *count=-1*, /)

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in B where subsection sub is found, such that sub is contained within B[start,end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in B where subsection sub is found, such that sub is contained within B[start,end]. Optional arguments start and end are interpreted as in slice notation.

Raise ValueError when the subsection is not found.

rjust(*width*, *fillchar*=b' ', /)

Return a right-justified string of length width.

Padding is done using the specified fill character.

rpartition(*sep*, /)

Partition the bytes into three parts using the given separator.

This will search for the separator sep in the bytes, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty bytes objects and the original bytes object.

rsplit(*sep*=None, *maxsplit*=-1)

Return a list of the sections in the bytes, using sep as the delimiter.

sep

The delimiter according which to split the bytes. None (the default value) means split on ASCII whitespace characters (space, tab, return, newline, formfeed, vertical tab).

maxsplit

Maximum number of splits to do. -1 (the default value) means no limit.

Splitting is done starting at the end of the bytes and working to the front.

rstrip(*bytes*=None, /)

Strip trailing bytes contained in the argument.

If the argument is omitted or None, strip trailing ASCII whitespace.

split(*sep*=None, *maxsplit*=-1)

Return a list of the sections in the bytes, using sep as the delimiter.

sep

The delimiter according which to split the bytes. None (the default value) means split on ASCII whitespace characters (space, tab, return, newline, formfeed, vertical tab).

maxsplit

Maximum number of splits to do. -1 (the default value) means no limit.

splitlines(*keepends*=False)

Return a list of the lines in the bytes, breaking at line boundaries.

Line breaks are not included in the resulting list unless keepends is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if B starts with the specified prefix, False otherwise. With optional start, test B beginning at that position. With optional end, stop comparing B at that position. *prefix* can also be a tuple of bytes to try.

strip(*bytes=None*, /)

Strip leading and trailing bytes contained in the argument.

If the argument is omitted or None, strip leading and trailing ASCII whitespace.

swapcase() → copy of B

Return a copy of B with uppercase ASCII characters converted to lowercase ASCII and vice versa.

title() → copy of B

Return a titlecased version of B, i.e. ASCII words start with uppercase characters, all remaining cased characters have lowercase.

translate(*table*, /, *delete=b''*)

Return a copy with each character mapped by the given translation table.

table

Translation table, which must be a bytes object of length 256.

All characters occurring in the optional argument *delete* are removed. The remaining characters are mapped through the given translation table.

upper() → copy of B

Return a copy of B with all ASCII characters converted to uppercase.

zfill(*width*, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The original string is never truncated.

volatility3.framework.symbols package

class SymbolSpace

Bases: *SymbolSpaceInterface*

Handles an ordered collection of SymbolTables.

This collection is ordered so that resolution of symbols can proceed down through the ranks if a namespace isn't specified.

class UnresolvedTemplate(*type_name*, ***kwargs*)

Bases: *ReferenceTemplate*

Class to highlight when missing symbols are present.

This class is identical to a reference template, but differentiable by its classname. It will output a debug log to indicate when it has been instantiated and with what name.

This class is designed to be output ONLY as part of the SymbolSpace resolution system. Individual SymbolTables that cannot resolve a symbol should still return a SymbolError to indicate this failure in resolution.

Stores the keyword arguments for later object creation.

child_template(*args, **kwargs)

Referenced symbols must be appropriately resolved before they can provide information such as size
This is because the size request has no context within which to determine the actual symbol structure.

Return type

Any

property children: *List[Template]*

The children of this template (such as member types, sub-types and base-types where they are relevant).

Used to traverse the template tree.

clone()

Returns a copy of the original Template as constructed (without *update_vol* additions having been made)

Return type

Template

has_member(*args, **kwargs)

Referenced symbols must be appropriately resolved before they can provide information such as size
This is because the size request has no context within which to determine the actual symbol structure.

Return type

Any

relative_child_offset(*args, **kwargs)

Referenced symbols must be appropriately resolved before they can provide information such as size
This is because the size request has no context within which to determine the actual symbol structure.

Return type

Any

replace_child(*args, **kwargs)

Referenced symbols must be appropriately resolved before they can provide information such as size
This is because the size request has no context within which to determine the actual symbol structure.

Return type

Any

property size: *Any*

Referenced symbols must be appropriately resolved before they can provide information such as size
This is because the size request has no context within which to determine the actual symbol structure.

update_vol(**new_arguments)

Updates the keyword arguments with values that will **not** be carried across to clones.

Return type

None

property vol: *ReadOnlyMapping*

Returns a volatility information object, much like the *ObjectInformation* provides.

append(value)

Adds a symbol_list to the end of the space.

Return type

None

clear_symbol_cache(table_name=None)

Clears the symbol cache for the specified table name. If no table name is specified, the caches of all symbol tables are cleared.

Return type`None`**free_table_name**(*prefix='layer'*)

Returns an unused table name to ensure no collision occurs when inserting a symbol table.

Return type`str`**get**(*k*, [*d*]) → *D*[*k*] if *k* in *D*, else *d*. *d* defaults to `None`.**get_enumeration**(*enum_name*)

Look-up a set of enumeration choices from a specific symbol table.

Return type`Template`**get_symbol**(*symbol_name*)

Look-up a symbol name across all the contained symbol spaces.

Return type`SymbolInterface`**get_symbols_by_location**(*offset*, *size=0*, *table_name=None*)

Returns all symbols that exist at a specific relative address.

Return type`Iterable[str]`**get_symbols_by_type**(*type_name*)

Returns all symbols based on the type of the symbol.

Return type`Iterable[str]`**get_type**(*type_name*)

Takes a symbol name and resolves it.

This method ensures that all referenced templates (including self-referential templates) are satisfied as `ObjectTemplates`

Return type`Template`**has_enumeration**(*name*)

Determines whether an enumeration choice exists in the contained symbol tables.

Return type`bool`**has_symbol**(*name*)

Determines whether a symbol exists in the contained symbol tables.

Return type`bool`**has_type**(*name*)

Determines whether a type exists in the contained symbol tables.

Return type`bool`

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

remove(*key*)

Removes a named symbol_list from the space.

Return type

None

values() → an object providing a view on D's values

class SymbolType(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: *Enum*

ENUM = 3

SYMBOL = 2

TYPE = 1

symbol_table_is_64bit(*context, symbol_table_name*)

Returns a boolean as to whether a particular symbol table within a context is 64-bit or not.

Return type

bool

Subpackages

volatility3.framework.symbols.generic package

class GenericIntelProcess(*context, type_name, object_info, size, members*)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod `has_member(template, member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**member**(*attr*='member')

Specifically named method for retrieving members.

Return type`object`**property vol:** `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

volatility3.framework.symbols.linux package**class** `LinuxKernelIntermedSymbols(*args, **kwargs)`

Bases: `IntermediateSymbolTable`

Instantiates a SymbolTable based on an IntermediateSymbolFormat JSON file. This is validated against the appropriate schema. The validation can be disabled by passing `validate = False`, but this should almost never be done.

Parameters

- **context** – The volatility context for the symbol table
- **config_path** – The configuration path for the symbol table
- **name** – The name for the symbol table (this is used in symbols e.g. table!symbol)
- **isf_url** – The URL pointing to the ISF file location
- **native_types** – The NativeSymbolTable that contains the native types for this symbol table
- **table_mapping** – A dictionary linking names referenced in the file with symbol tables in the context
- **validate** – Determines whether the ISF file will be validated against the appropriate schema
- **class_types** – A dictionary of type names and classes that override StructType when they are instantiated
- **symbol_mask** – An address mask used for all returned symbol offsets from this table (a mask of 0 disables masking)

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type`HierarchicalDict`**clear_symbol_cache**(**args, **kwargs*)

Clears the symbol cache of this symbol table.

property config: [*HierarchicalDict*](#)

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: [*str*](#)

The configuration path on which this configurable lives.

property context: [*ContextInterface*](#)

The context object that this configurable belongs to/configuration is stored in.

classmethod create(*context, config_path, sub_path, filename, native_types=None, table_mapping=None, class_types=None, symbol_mask=0*)

Takes a context and loads an intermediate symbol table based on a filename.

Parameters

- **context** ([*ContextInterface*](#)) – The context that the current plugin is being run within
- **config_path** ([*str*](#)) – The configuration path for reading/storing configuration information this symbol table may use
- **sub_path** ([*str*](#)) – The path under a suitable symbol path (defaults to volatility3/symbols and volatility3/framework/symbols) to check
- **filename** ([*str*](#)) – Basename of the file to find under the sub_path
- **native_types** ([*Optional\[NativeTableInterface\]*](#)) – Set of native types, defaults to native types read from the intermediate symbol format file
- **table_mapping** ([*Optional\[Dict\[str, str\]\]*](#)) – a dictionary of table names mentioned within the ISF file, and the tables within the context which they map to
- **symbol_mask** ([*int*](#)) – An address mask used for all returned symbol offsets from this table (a mask of 0 disables masking)

Return type

[*str*](#)

Returns

the name of the added symbol table

del_type_class(**args, **kwargs*)

Removes the associated class override for a specific Symbol type.

property enumerations

Returns an iterator of the Enumeration names.

classmethod file_symbol_url(*sub_path, filename=None*)

Returns an iterator of appropriate file-scheme symbol URLs that can be opened by a ResourceAccessor class.

Filter reduces the number of results returned to only those URLs containing that string

Return type

[*Generator\[str, None, None\]*](#)

get_enumeration(**args, **kwargs*)

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

[*List\[RequirementInterface\]*](#)

get_symbol(*args, **kwargs)

Resolves a symbol name into a symbol object.

If the symbol isn't found, it raises a SymbolError exception

get_symbol_type(name)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type

`Optional[Template]`

get_symbols_by_location(offset, size=0)

Returns the name of all symbols in this table that live at a particular offset.

Return type

`Iterable[str]`

get_symbols_by_type(type_name)

Returns the name of all symbols in this table that have type matching type_name.

Return type

`Iterable[str]`

get_type(*args, **kwargs)

Resolves a symbol name into an object template.

If the symbol isn't found it raises a SymbolError exception

get_type_class(*args, **kwargs)

Returns the class associated with a Symbol type.

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property metadata

property natives: `NativeTableInterface`

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(name, clazz)

Calls the set_type_class function but does not throw an exception. Returns whether setting the type class was successful. :type name: `str` :param name: The name of the type to override the class for :type clazz: `Type[ObjectInterface]` :param clazz: The actual class to override for the provided type name

Return type

`bool`

```
provides = {'type': 'interface'}
```

```
set_type_class(*args, **kwargs)
```

Overrides the object class for a specific Symbol type.

Name *must* be present in self.types

Parameters

- **name** – The name of the type to override the class for
- **clazz** – The actual class to override for the provided type name

property symbols

Returns an iterator of the Symbol names.

property types

Returns an iterator of the Symbol type names.

```
classmethod unsatisfied(context, config_path)
```

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

```
class LinuxUtilities(*args, **kwargs)
```

Bases: `VersionableInterface`

Class with multiple useful linux functions.

```
classmethod container_of(addr, type_name, member_name, vmlinux)
```

Cast a member of a structure out to the containing structure. It mimicks the Linux kernel macro `container_of()` see `include/linux/kernel.h`

Parameters

- **addr** (`int`) – The pointer to the member.
- **type_name** (`str`) – The type of the container struct this is embedded in.
- **member_name** (`str`) – The name of the member within the struct.
- **vmlinux** (`ModuleInterface`) – The kernel symbols object

Return type

`Optional[ObjectInterface]`

Returns

The constructed object or None

```
classmethod do_get_path(rdentry, rmnt, dentry, vsmnt)
```

Returns a pathname of the mount point or file It mimics the Linux kernel `prepend_path` function.

Parameters

- **rdentry** (`dentry *`) – A pointer to the root dentry

- **rmnt** (*vfsmount* *) – A pointer to the root vfsmount
- **dentry** (*dentry* *) – A pointer to the dentry
- **vfsmnt** (*vfsmount* *) – A pointer to the vfsmount

Returns

Pathname of the mount point or file

Return type

`str`

classmethod `files_descriptors_for_process(context, symbol_table, task)`

classmethod `generate_kernel_handler_info(context, kernel_module_name, mods_list)`

A helper function that gets the beginning and end address of the kernel module

Return type

`List[Tuple[str, int, int]]`

classmethod `get_module_from_volobj_type(context, volobj)`

Get the vmlinux from a vol obj

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **volobj** (*vol object*) – A vol object

Raises

ValueError – If it cannot obtain any module from the symbol table

Return type

ModuleInterface

Returns

A kernel object (vmlinux)

classmethod `get_path_mnt(task, mnt)`

Returns the mount point pathname relative to the task's root directory.

Parameters

- **task** (*task_struct*) – A reference task
- **mnt** (*vfsmount* or *mount*) – A mounted filesystem or a mount point. - kernels < 3.3.8 type is 'vfsmount' - kernels >= 3.3.8 type is 'mount'

Returns

Pathname of the mount point relative to the task's root directory.

Return type

`str`

classmethod `lookup_module_address(kernel_module, handlers, target_address)`

Searches between the start and end address of the kernel module using target_address. Returns the module and symbol name of the address provided.

classmethod `mask_mods_list(context, layer_name, mods)`

A helper function to mask the starting and end address of kernel modules

Return type

`List[Tuple[str, int, int]]`

classmethod `path_for_file(context, task, filp)`

Returns a file (or sock pipe) pathname relative to the task's root directory.

A 'file' structure doesn't have enough information to properly restore its full path we need the root mount information from `task_struct` to determine this

Parameters

- **context** – The context to retrieve required elements (layers, symbol tables) from
- **task** (`task_struct`) – A reference task
- **filp** (`file *`) – A pointer to an open file

Returns

A file (or sock pipe) pathname relative to the task's root directory.

Return type

`str`

version = (2, 1, 0)

classmethod `walk_internal_list(vmlinux, struct_name, list_member, list_start)`

Subpackages

`volatility3.framework.symbols.linux.extensions` package

class `bpf_prog(context, type_name, object_info, size, members)`

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`

Bases: `VolTemplateProxy`

classmethod `child_template(template, child)`

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children(template)`

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member(template, member_name)`

Returns whether the object would contain a member called `member_name`.

Return type

`bool`

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

get_type()

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

`object`

property vol: `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class `bt_sock`(*context, type_name, object_info, size, members*)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`

Bases: `VolTemplateProxy`

classmethod `child_template`(*template, child*)

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children`(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member`(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset`(*template, child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child`(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size`(*template*)

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_protocol()

get_state()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

str

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class `cred`(*context, type_name, object_info, size, members*)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`

Bases: `VolTemplateProxy`

classmethod `child_template`(*template, child*)

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children`(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member`(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset`(*template, child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child`(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size`(*template*)

Method to return the size of this type.

Return type

`int`

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

property `euid`

Returns the effective user ID

Returns

the effective user ID value

Return type`int`**get_symbol_table_name()**

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type`str`**has_member(member_name)**

Returns whether the object would contain a member called member_name.

Return type`bool`**has_valid_member(member_name)**

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type`bool`**has_valid_members(member_names)**

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**member(attr='member')**

Specifically named method for retrieving members.

Return type`object`**property vol: *ReadOnlyMapping***

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

class dentry(context, type_name, object_info, size, members)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object

- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

d_ancestor(*ancestor_dentry*)

Search for an ancestor

Returns the ancestor dentry which is a child of “ancestor_dentry”, if “ancestor_dentry” is an ancestor of “child_dentry”, else None.

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type`str`**has_member**(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type`bool`**has_valid_member**(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type`bool`**has_valid_members**(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**is_root**()**Return type**`bool`**is_subdir**(*old_dentry*)

Is this dentry a subdirectory of *old_dentry*?

Returns true if this dentry is a subdirectory of the parent (at any depth). Otherwise, it returns false.

member(*attr*='member')

Specifically named method for retrieving members.

Return type`object`**path**()

Based on `__dentry_path` Linux kernel function

Return type`str`**property** **vol**: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class **files_struct**(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_fds()

Return type

ObjectInterface

get_max_fds()

Return type

ObjectInterface

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(attr='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

class fs_struct(context, type_name, object_info, size, members)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template*, *child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template*, *member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template*, *old_child*, *new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

ObjectInterface

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_root_dentry()

get_root_mnt()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

str

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class inet_sock(*context, type_name, object_info, size, members*)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod size(*template*)

Method to return the size of this type.

Return type

`int`

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_dst_addr()

get_dst_port()

get_family()

get_protocol()

get_src_addr()

get_src_port()

get_state()

Return a string representing the sock state.

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class kernel_cap_struct(*context, type_name, object_info, size, members*)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod **children**(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod **has_member**(*template*, *member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

classmethod **relative_child_offset**(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod **replace_child**(*template*, *old_child*, *new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod **size**(*template*)

Method to return the size of this type.

Return type

`int`

classmethod **capabilities_to_string**(*capabilities_bitfield*)

Translates a capability bitfield to a list of capability strings.

Parameters

capabilities_bitfield (`int`) – The capability bitfield value.

Returns

A list of capability strings.

Return type

`List[str]`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

enumerate_capabilities()

Returns the list of capability strings.

Returns

The list of capability strings.

Return type

`List[str]`

get_capabilities()

Returns the capability bitfield value

Returns

The capability bitfield value.

Return type`int`**get_kernel_cap_full()**

Return the maximum value allowed for this kernel for a capability

Returns

The capability full bitfield mask

Return type`int`**classmethod get_last_cap_value()**

Returns the latest capability ID supported by the framework.

Returns

The latest capability ID supported by the framework.

Return type`int`**get_symbol_table_name()**

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type`str`**has_capability(capability)**

Checks if the given capability string is enabled.

Parameters

capability (`str`) – A string representing the capability i.e. `dac_read_search`

Raises

AttributeError – If the given capability is unknown to the framework.

Returns

“True” if the given capability is enabled.

Return type`bool`**has_member(member_name)**

Returns whether the object would contain a member called member_name.

Return type`bool`**has_valid_member(member_name)**

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class kernel_cap_t(*context, type_name, object_info, size, members*)

Bases: *kernel_cap_struct*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

classmethod `capabilities_to_string(capabilities_bitfield)`

Translates a capability bitfield to a list of capability strings.

Parameters

capabilities_bitfield (`int`) – The capability bitfield value.

Returns

A list of capability strings.

Return type

`List[str]`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

enumerate_capabilities()

Returns the list of capability strings.

Returns

The list of capability strings.

Return type

`List[str]`

get_capabilities()

Returns the capability bitfield value

Returns

The capability bitfield value.

Return type

`int`

get_kernel_cap_full()

Return the maximum value allowed for this kernel for a capability

Returns

The capability full bitfield mask

Return type

`int`

classmethod `get_last_cap_value()`

Returns the latest capability ID supported by the framework.

Returns

The latest capability ID supported by the framework.

Return type

`int`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_capability(capability)

Checks if the given capability string is enabled.

Parameters

capability (`str`) – A string representing the capability i.e. `dac_read_search`

Raises

AttributeError – If the given capability is unknown to the framework.

Returns

“True” if the given capability is enabled.

Return type

`bool`

has_member(member_name)

Returns whether the object would contain a member called `member_name`.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in `member_names`

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(attr='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class kobject(*context, type_name, object_info, size, members*)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

ObjectInterface

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

`object`

reference_count()**property** vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class list_head(*context, type_name, object_info, size, members*)

Bases: *StructType, Iterable*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object

- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template*, *child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template*, *member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template*, *old_child*, *new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

str

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type`bool`**has_valid_member**(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type`bool`**has_valid_members**(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**member**(*attr*='member')

Specifically named method for retrieving members.

Return type`object`**to_list**(*symbol_type*, *member*, *forward*=`True`, *sentinel*=`True`, *layer*=`None`)

Returns an iterator of the entries in the list.

Parameters

- **symbol_type** (`str`) – Type of the list elements
- **member** (`str`) – Name of the list_head member in the list elements
- **forward** (`bool`) – Set false to go backwards
- **sentinel** (`bool`) – Whether self is a “sentinel node”, meaning it is not embedded in a member of the list
- **https** (*Sentinel nodes are NOT yielded. See* [//en.wikipedia.org/wiki/Sentinel_node](https://en.wikipedia.org/wiki/Sentinel_node) for further reference) –
- **layer** (`Optional[str]`) – Name of layer to read from

Yields

Objects of the type specified via the “symbol_type” argument.

Return type`Iterator[ObjectInterface]`**property vol:** `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class maple_tree(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

MAPLE_ARANGE_64 = 3

MAPLE_DENSE = 0

MAPLE_LEAF_64 = 1

MAPLE_NODE_POINTER_MASK = 255

MAPLE_NODE_TYPE_MASK = 15

MAPLE_NODE_TYPE_SHIFT = 3

MAPLE_RANGE_64 = 2

MT_FLAGS_HEIGHT_MASK = 124

MT_FLAGS_HEIGHT_OFFSET = 2

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_slot_iter()

Parse the Maple Tree and return every non zero slot.

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

str

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class `mm_struct`(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`

Bases: `VolTemplateProxy`

classmethod `child_template`(*template*, *child*)

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children`(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member`(*template*, *member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset`(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child`(*template*, *old_child*, *new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size`(*template*)

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_maple_tree_iter()

Returns an iterator for the mm_mt member of an mm_struct.

Return type

`Iterable[ObjectInterface]`

get_mmap_iter()

Returns an iterator for the mmap list member of an mm_struct.

Return type

`Iterable[ObjectInterface]`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

get_vma_iter()

Returns an iterator for the VMAs in an mm_struct. Automatically choosing the mmap or mm_mt as required.

Return type

`Iterable[ObjectInterface]`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(attr='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

class `mnt_namespace`(*context, type_name, object_info, size, members*)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`

Bases: `VolTemplateProxy`

classmethod `child_template`(*template, child*)

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children`(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member`(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset`(*template, child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child`(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size`(*template*)

Method to return the size of this type.

Return type

`int`

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

`get_inode()`

`get_mount_points()`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(attr='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

class module(context, type_name, object_info, size, members)

Bases: *GenericIntelProcess*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxyBases: *VolTemplateProxy***classmethod child_template**(*template, child*)

Returns the template of a child to its parent.

Return type*Template***classmethod children**(*template*)

Method to list children of a template.

Return type*List[Template]***classmethod has_member**(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type*bool***classmethod relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type*int***classmethod replace_child**(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type*None***classmethod size**(*template*)

Method to return the size of this type.

Return type*int***cast**(*new_type_name, **additional*)Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_core_size()**get_elf_table_name**()**get_init_size**()**get_module_base**()**get_module_core**()**get_module_init**()**get_name**()

Get the name of the module as a string

get_sections()

Get sections of the module

get_symbol(*wanted_sym_name*)

Get symbol value for a given symbol name

get_symbol_by_address(*wanted_sym_address*)

Get symbol name for a given symbol address

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

get_symbols()

Get symbols of the module

Yields

A symbol object

get_symbols_names_and_addresses()

Get names and addresses for each symbol of the module

Yields

A tuple for each symbol containing the symbol name and its corresponding value

Return type

`Tuple[str, int]`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

`object`

property num_symtab

property section_strtab

property section_symtab

property vol: [ReadOnlyMapping](#)

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

class mount(context, type_name, object_info, size, members)

Bases: [StructType](#)

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** ([ContextInterface](#)) – The context associated with the object
- **type_name** ([str](#)) – The name of the type structure for the object
- **object_info** ([ObjectInformation](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

```
MNT_FLAGS = {1: 'nosuid', 2: 'nodev', 4: 'noexec', 8: 'noatime', 16:
'nodiratime', 32: 'relatime'}
```

```
MNT_NOATIME = 8
```

```
MNT_NODEV = 2
```

```
MNT_NODIRATIME = 16
```

```
MNT_NOEXEC = 4
```

```
MNT_NOSUID = 1
```

```
MNT_READONLY = 64
```

```
MNT_RELATIME = 32
```

```
MNT_SHARED = 4096
```

```
MNT_SHRINKABLE = 256
```

```
MNT_UNBINDABLE = 8192
```

```
MNT_WRITE_HOLD = 512
```

class VolTemplateProxy

Bases: [VolTemplateProxy](#)

classmethod child_template(template, child)

Returns the template of a child to its parent.

Return type

[Template](#)

classmethod `children(template)`

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member(template, member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_dentry_current()

Returns the root of the mounted tree

Returns

A dentry pointer

get_dentry_parent()

Returns the parent root of the mounted tree

Returns

A dentry pointer

get_devname()

Return type

`str`

get_dominating_id(root)

Get ID of closest dominating peer group having a representative under the given root.

Return type

`int`

get_flags_access()

Return type

`str`

get_flags_opts()

Return type
`Iterable[str]`

get_mnt_flags()

get_mnt_mountpoint()

Gets the dentry of the mountpoint

Returns
 A dentry pointer

get_mnt_parent()

Gets the fs where we are mounted on

Returns
 A mount pointer

get_mnt_root()

get_mnt_sb()

get_parent_mount()

get_peer_under_root(*ns, root*)

Return true if path is reachable from root. It mimics the kernel function `is_path_reachable()`, ref: `fs/namespace.c`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the `table_name` is not valid within the object’s context

Return type
`str`

get_vfsmnt_current()

Returns the fs where we are mounted on

Returns
 A ‘vfsmount’

get_vfsmnt_parent()

Gets the parent fs (vfsmount) to where it’s mounted on

Returns
 A ‘vfsmount’

has_member(*member_name*)

Returns whether the object would contain a member called `member_name`.

Return type
`bool`

has_parent()

Checks if this mount has a parent

Returns

‘True’ if this mount has a parent

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

is_path_reachable(*current_dentry*, *root*)

Return true if path is reachable. It mimics the kernel function with same name, ref fs/namespace.c:

is_shared()**Return type**

`bool`

is_slave()**Return type**

`bool`

is_unbindable()**Return type**

`bool`

member(*attr*=‘member’)

Specifically named method for retrieving members.

Return type

`object`

next_peer()**property vol:** *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class `net(context, type_name, object_info, size, members)`

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`

Bases: `VolTemplateProxy`

classmethod `child_template(template, child)`

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children(template)`

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member(template, member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(`new_type_name, **additional`)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

`get_inode()`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(attr='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

class netlink_sock(context, type_name, object_info, size, members)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

ObjectInterface

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_dst_portid()

get_portid()

get_protocol()

get_state()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

str

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class packet_sock(*context, type_name, object_info, size, members*)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: `VolTemplateProxy`

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

`Template`

classmethod children(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod size(*template*)

Method to return the size of this type.

Return type

`int`

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_protocol()

get_state()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the *table_name* is not valid within the object’s context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class qstr(*context, type_name, object_info, size, members*)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

bool

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: [*ObjectInterface*](#)

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

name_as_str()

Return type

`str`

property vol: `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class sock(*context, type_name, object_info, size, members*)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: `VolTemplateProxy`

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

`Template`

classmethod children(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_family()

get_inode()

get_protocol()

get_state()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

get_type()

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

property *vol*: `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class `socket`(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`

Bases: `VolTemplateProxy`

classmethod `child_template`(*template*, *child*)

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children`(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member`(*template*, *member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset`(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child`(*template*, *old_child*, *new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size`(*template*)

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_inode()

get_state()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

str

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class struct_file(*context, type_name, object_info, size, members*)

Bases: [StructType](#)

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** ([ContextInterface](#)) – The context associated with the object
- **type_name** ([str](#)) – The name of the type structure for the object
- **object_info** ([ObjectInformation](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: [VolTemplateProxy](#)

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

[Template](#)

classmethod children(*template*)

Method to list children of a template.

Return type

[List](#)[[Template](#)]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

[bool](#)

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

[int](#)

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

[None](#)

classmethod size(*template*)

Method to return the size of this type.

Return type

[int](#)

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

[ObjectInterface](#)

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_dentry()

Return type

[ObjectInterface](#)

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

get_vfsmnt()

Returns the fs (vfsmount) where this file is mounted

Return type

`ObjectInterface`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(attr='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

class super_block(context, type_name, object_info, size, members)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object

- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

MINORBITS = 20

SB_DIRSYNC = 128

SB_I_VERSION = 8388608

SB_KERNMOUNT = 4194304

SB_LAZYTIME = 33554432

SB_MANDLOCK = 64

SB_NOATIME = 1024

SB_NODEV = 4

SB_NODIRATIME = 2048

SB_NOEXEC = 8

SB_NOSUID = 2

SB_OPTS = {16: 'sync', 64: 'mand', 128: 'dirsync', 33554432: 'lazytime'}

SB_POSIXACL = 65536

SB_RDONLY = 1

SB_SILENT = 32768

SB_SYNCHRONOUS = 16

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_flags_access()

Return type

`str`

get_flags_opts()

Return type

`Iterable[str]`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

get_type()

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

property major: *int*

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property minor: *int*

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class task_struct(*context, type_name, object_info, size, members*)

Bases: *GenericIntelProcess*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

add_process_layer(*config_prefix=None, preferred_name=None*)

Constructs a new layer based on the process's DTB.

Returns the name of the Layer or None.

Return type

`Optional[str]`

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_process_memory_sections(*heap_only=False*)

Returns a list of sections based on the memory manager's view of this task's virtual memory.

Return type

`Generator[Tuple[int, int], None, None]`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

get_threads()

Returns a list of the task_struct based on the list_head thread_node structure.

Return type

`Iterable[ObjectInterface]`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type`bool`**has_valid_members**(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**property is_kernel_thread:** `bool`

Checks if this task is a kernel thread.

Returns

True, if this task is a kernel thread. Otherwise, False.

Return type`bool`**property is_thread_group_leader:** `bool`

Checks if this task is a thread group leader.

Returns

True, if this task is a thread group leader. Otherwise, False.

Return type`bool`**property is_user_thread:** `bool`

Checks if this task is a user thread.

Returns

True, if this task is a user thread. Otherwise, False.

Return type`bool`**member**(*attr*='member')

Specifically named method for retrieving members.

Return type`object`**property vol:** `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class unix_sock(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object

- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_inode()

get_name()

get_protocol()

get_state()

Return a string representing the sock state.

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table

- **KeyError** – If the `table_name` is not valid within the object's context

Return type`str`**has_member**(*member_name*)

Returns whether the object would contain a member called `member_name`.

Return type`bool`**has_valid_member**(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type`bool`**has_valid_members**(*member_names*)

Returns whether the object has all of the members listed in `member_names`

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**member**(*attr*='member')

Specifically named method for retrieving members.

Return type`object`**property** `vol: ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class **vfsmount**(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `StructType`

Constructs an Object adhering to the `ObjectInterface`.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class **VolTemplateProxy**

Bases: `VolTemplateProxy`

classmethod **child_template**(*template*, *child*)

Returns the template of a child to its parent.

Return type*Template***classmethod children**(*template*)

Method to list children of a template.

Return type*List[Template]***classmethod has_member**(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type*bool***classmethod relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type*int***classmethod replace_child**(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type*None***classmethod size**(*template*)

Method to return the size of this type.

Return type*int***cast**(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

ObjectInterface

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_dentry_current()

Returns the root of the mounted tree

Returns

A dentry pointer

get_dentry_parent()

Returns the parent root of the mounted tree

Returns

A dentry pointer

get_devname()**Return type***str***get_flags_access**()**Return type***str*

get_flags_opts()

Return type

`Iterable[str]`

get_mnt_flags()

get_mnt_mountpoint()

Gets the dentry of the mountpoint

Returns

A dentry pointer

get_mnt_parent()

Gets the mnt_parent member.

Returns

A vfsmount pointer For kernels \geq 3.3.8: A mount pointer

Return type

For kernels $<$ 3.3.8

get_mnt_root()

get_mnt_sb()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

get_vfsmnt_current()

Returns the current fs where we are mounted on

Returns

A vfsmount pointer

get_vfsmnt_parent()

Gets the parent fs (vfsmount) to where it’s mounted on

Returns

A vfsmount pointer For kernels \geq 3.3.8: A vfsmount object

Return type

For kernels $<$ 3.3.8

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_parent()

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

is_equal(*vfsmount_ptr*)

Helper to make sure it is comparing two pointers to ‘vfsmount’.

Depending on the kernel version, the calling object (self) could be a ‘vfsmount *’ (<3.3.8) or a ‘vfsmount’ (>=3.3.8). This way we trust in the framework “auto” dereferencing ability to assure that when we reach this point ‘self’ will be a ‘vfsmount’ already and self.vol.offset a ‘vfsmount *’ and not a ‘vfsmount **’. The argument must be a ‘vfsmount *’. Typically, it’s called from do_get_path().

Parameters

vfsmount_ptr (*vfsmount **) – A pointer to a ‘vfsmount’

Raises

exceptions.VolatilityException – If vfsmount_ptr is not a ‘vfsmount *’

Returns

‘True’ if the given argument points to the the same ‘vfsmount’ as ‘self’.

Return type

bool

is_shared()

Return type

bool

is_slave()

Return type

bool

is_unbindable()

Return type

bool

is_valid()

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: [*ReadOnlyMapping*](#)

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class vm_area_struct(*context, type_name, object_info, size, members*)

Bases: [*StructType*](#)

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** ([*ContextInterface*](#)) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** ([*ObjectInformation*](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: [*VolTemplateProxy*](#)

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

[*Template*](#)

classmethod children(*template*)

Method to list children of a template.

Return type

[*List\[Template\]*](#)

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

[*bool*](#)

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

[*int*](#)

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

[*None*](#)

classmethod size(*template*)

Method to return the size of this type.

Return type

[*int*](#)

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

[*ObjectInterface*](#)

Note: If new type name does not include a symbol table, the symbol table for the current object is used

```
extended_flags = {1: 'VM_READ', 2: 'VM_WRITE', 4: 'VM_EXEC', 8: 'VM_SHARED', 16:
'VM_MAYREAD', 32: 'VM_MAYWRITE', 64: 'VM_MAYEXEC', 128: 'VM_MAYSHARE', 256:
'VM_GROWSDOWN', 512: 'VM_NOHUGEPAGE', 1024: 'VM_PFNMAP', 2048: 'VM_DENYWRITE',
4096: 'VM_EXECUTABLE', 8192: 'VM_LOCKED', 16384: 'VM_IO', 32768: 'VM_SEQ_READ',
65536: 'VM_RAND_READ', 131072: 'VM_DONTCOPY', 262144: 'VM_DONTEXPAND', 524288:
'VM_RESERVED', 1048576: 'VM_ACCOUNT', 2097152: 'VM_NORESERVE', 4194304:
'VM_HUGETLB', 8388608: 'VM_NONLINEAR', 16777216: 'VM_MAPPED_COPY_HUGEPAGE',
33554432: 'VM_INSERTPAGE', 67108864: 'VM_ALWAYSDUMP', 134217728:
'VM_CAN_NONLINEAR', 268435456: 'VM_MIXEDMAP', 536870912: 'VM_SAO', 1073741824:
'VM_PFN_AT_MMAP', 2147483648: 'VM_MERGEABLE'}
```

`get_flags()`

Return type

`str`

`get_name(context, task)`

`get_page_offset()`

Return type

`int`

`get_protection()`

Return type

`str`

`get_symbol_table_name()`

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

`has_member(member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

`has_valid_member(member_name)`

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

`has_valid_members(member_names)`

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**is_suspicious**(*proclayer=None*)**member**(*attr='member'*)

Specifically named method for retrieving members.

Return type`object`**perm_flags** = {1: 'r', 2: 'w', 4: 'x'}**property** vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class **vsock_sock**(*context, type_name, object_info, size, members*)Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class **VolTemplateProxy**Bases: *VolTemplateProxy***classmethod** **child_template**(*template, child*)

Returns the template of a child to its parent.

Return type*Template***classmethod** **children**(*template*)

Method to list children of a template.

Return type*List[Template]***classmethod** **has_member**(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type`bool`**classmethod** **relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type`int`**classmethod** **replace_child**(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_protocol()

get_state()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

`object`

property vol: [*ReadOnlyMapping*](#)

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class xdp_sock(*context, type_name, object_info, size, members*)

Bases: [*StructType*](#)

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** ([*ContextInterface*](#)) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** ([*ObjectInformation*](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: [*VolTemplateProxy*](#)

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

[*Template*](#)

classmethod children(*template*)

Method to list children of a template.

Return type

[*List\[Template\]*](#)

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

[*bool*](#)

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

[*int*](#)

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

[*None*](#)

classmethod size(*template*)

Method to return the size of this type.

Return type

[*int*](#)

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

[*ObjectInterface*](#)

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_protocol()

get_state()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(attr='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

Submodules

volatility3.framework.symbols.linux.extensions.bash module

class `hist_entry`(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`

Bases: `VolTemplateProxy`

classmethod `child_template`(*template*, *child*)

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children`(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member`(*template*, *member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset`(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child`(*template*, *old_child*, *new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size`(*template*)

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_command()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

get_time_as_integer()

get_time_object()

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

is_valid()

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

volatility3.framework.symbols.linux.extensions.elf module**class** `elf(context, type_name, object_info, size, members)`Bases: `StructType`

Class used to create elf objects. It overrides the typename to `Elf32_` or `Elf64_`, depending on the corresponding value on `e_ident`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`Bases: `VolTemplateProxy`**classmethod** `child_template(template, child)`

Returns the template of a child to its parent.

Return type`Template`**classmethod** `children(template)`

Method to list children of a template.

Return type`List[Template]`**classmethod** `has_member(template, member_name)`

Returns whether the object would contain a member called member_name.

Return type`bool`**classmethod** `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type`int`**classmethod** `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type`None`**classmethod** `size(template)`

Method to return the size of this type.

Return type`int`**cast**(`new_type_name, **additional`)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_program_headers()

get_section_headers()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

get_symbols()

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

is_valid()

Determine whether it is a valid object

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class `elf_phdr(*args, **kwargs)`

Bases: `StructType`

An elf program header

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** – The context associated with the object
- **type_name** – The name of the type structure for the object
- **object_info** – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`

Bases: `VolTemplateProxy`

classmethod `child_template(template, child)`

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children(template)`

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member(template, member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(`new_type_name, **additional`)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

dynamic_sections()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

get_vaddr()**has_member(member_name)**

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(attr='member')

Specifically named method for retrieving members.

Return type

`object`

property parent_e_type**property parent_offset****property type_prefix****property vol:** *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

class elf_sym(*args, **kwargs)

Bases: *StructType*

An elf symbol entry

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** – The context associated with the object
- **type_name** – The name of the type structure for the object
- **object_info** – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

property cached_strtab**cast**(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_name()**get_symbol_table_name**()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table

- **KeyError** – If the `table_name` is not valid within the object's context

Return type`str`**has_member**(*member_name*)Returns whether the object would contain a member called `member_name`.**Return type**`bool`**has_valid_member**(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters**member_name** (`str`) – Name of the member to test access to determine if the member is valid or not**Return type**`bool`**has_valid_members**(*member_names*)Returns whether the object has all of the members listed in `member_names`**Parameters****member_names** (`List[str]`) – List of names to test as to members with those names validity**Return type**`bool`**member**(*attr*='member')

Specifically named method for retrieving members.

Return type`object`**property vol:** [*ReadOnlyMapping*](#)

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

Submodules

volatility3.framework.symbols.linux.bash module

class BashIntermedSymbols(*args, **kwargs)Bases: [*IntermediateSymbolTable*](#)

Instantiates a SymbolTable based on an IntermediateSymbolFormat JSON file. This is validated against the appropriate schema. The validation can be disabled by passing `validate = False`, but this should almost never be done.

Parameters

- **context** – The volatility context for the symbol table
- **config_path** – The configuration path for the symbol table
- **name** – The name for the symbol table (this is used in symbols e.g. `table!symbol`)

- **isf_url** – The URL pointing to the ISF file location
- **native_types** – The NativeSymbolTable that contains the native types for this symbol table
- **table_mapping** – A dictionary linking names referenced in the file with symbol tables in the context
- **validate** – Determines whether the ISF file will be validated against the appropriate schema
- **class_types** – A dictionary of type names and classes that override StructType when they are instantiated
- **symbol_mask** – An address mask used for all returned symbol offsets from this table (a mask of 0 disables masking)

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

clear_symbol_cache(*args, **kwargs)

Clears the symbol cache of this symbol table.

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod create(context, config_path, sub_path, filename, native_types=None, table_mapping=None, class_types=None, symbol_mask=0)

Takes a context and loads an intermediate symbol table based on a filename.

Parameters

- **context** (*ContextInterface*) – The context that the current plugin is being run within
- **config_path** (*str*) – The configuration path for reading/storing configuration information this symbol table may use
- **sub_path** (*str*) – The path under a suitable symbol path (defaults to volatility3/symbols and volatility3/framework/symbols) to check
- **filename** (*str*) – Basename of the file to find under the sub_path
- **native_types** (*Optional[NativeTableInterface]*) – Set of native types, defaults to native types read from the intermediate symbol format file
- **table_mapping** (*Optional[Dict[str, str]]*) – a dictionary of table names mentioned within the ISF file, and the tables within the context which they map to
- **symbol_mask** (*int*) – An address mask used for all returned symbol offsets from this table (a mask of 0 disables masking)

Return type

str

Returns

the name of the added symbol table

del_type_class(*args, **kwargs)

Removes the associated class override for a specific Symbol type.

property enumerations

Returns an iterator of the Enumeration names.

classmethod file_symbol_url(sub_path, filename=None)

Returns an iterator of appropriate file-scheme symbol URLs that can be opened by a ResourceAccessor class.

Filter reduces the number of results returned to only those URLs containing that string

Return type

Generator[str, None, None]

get_enumeration(*args, **kwargs)

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

List[RequirementInterface]

get_symbol(*args, **kwargs)

Resolves a symbol name into a symbol object.

If the symbol isn't found, it raises a SymbolError exception

get_symbol_type(name)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type

Optional[Template]

get_symbols_by_location(offset, size=0)

Returns the name of all symbols in this table that live at a particular offset.

Return type

Iterable[str]

get_symbols_by_type(type_name)

Returns the name of all symbols in this table that have type matching type_name.

Return type

Iterable[str]

get_type(*args, **kwargs)

Resolves a symbol name into an object template.

If the symbol isn't found it raises a SymbolError exception

get_type_class(*args, **kwargs)

Returns the class associated with a Symbol type.

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property metadata**property natives:** *NativeTableInterface*

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name, clazz*)

Calls the `set_type_class` function but does not throw an exception. Returns whether setting the type class was successful. :type name: *str* :param name: The name of the type to override the class for :type clazz: *Type[ObjectInterface]* :param clazz: The actual class to override for the provided type name

Return type

bool

set_type_class(*args, **kwargs)

Overrides the object class for a specific Symbol type.

Name *must* be present in `self.types`

Parameters

- **name** – The name of the type to override the class for
- **clazz** – The actual class to override for the provided type name

property symbols

Returns an iterator of the Symbol names.

property types

Returns an iterator of the Symbol type names.

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

volatility3.framework.symbols.mac package

class `MacKernelIntermedSymbols(*args, **kwargs)`

Bases: [`IntermediateSymbolTable`](#)

Instantiates a SymbolTable based on an IntermediateSymbolFormat JSON file. This is validated against the appropriate schema. The validation can be disabled by passing `validate = False`, but this should almost never be done.

Parameters

- **context** – The volatility context for the symbol table
- **config_path** – The configuration path for the symbol table
- **name** – The name for the symbol table (this is used in symbols e.g. `table!symbol`)
- **isf_url** – The URL pointing to the ISF file location
- **native_types** – The NativeSymbolTable that contains the native types for this symbol table
- **table_mapping** – A dictionary linking names referenced in the file with symbol tables in the context
- **validate** – Determines whether the ISF file will be validated against the appropriate schema
- **class_types** – A dictionary of type names and classes that override StructType when they are instantiated
- **symbol_mask** – An address mask used for all returned symbol offsets from this table (a mask of 0 disables masking)

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

[`HierarchicalDict`](#)

clear_symbol_cache(*args, **kwargs)

Clears the symbol cache of this symbol table.

property config: [`HierarchicalDict`](#)

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: [`ContextInterface`](#)

The context object that this configurable belongs to/configuration is stored in.

classmethod create(*context, config_path, sub_path, filename, native_types=None, table_mapping=None, class_types=None, symbol_mask=0*)

Takes a context and loads an intermediate symbol table based on a filename.

Parameters

- **context** ([`ContextInterface`](#)) – The context that the current plugin is being run within

- **config_path** (*str*) – The configuration path for reading/storing configuration information this symbol table may use
- **sub_path** (*str*) – The path under a suitable symbol path (defaults to volatility3/symbols and volatility3/framework/symbols) to check
- **filename** (*str*) – Basename of the file to find under the sub_path
- **native_types** (*Optional[NativeTableInterface]*) – Set of native types, defaults to native types read from the intermediate symbol format file
- **table_mapping** (*Optional[Dict[str, str]]*) – a dictionary of table names mentioned within the ISF file, and the tables within the context which they map to
- **symbol_mask** (*int*) – An address mask used for all returned symbol offsets from this table (a mask of 0 disables masking)

Return type*str***Returns**

the name of the added symbol table

del_type_class(**args, **kwargs*)

Removes the associated class override for a specific Symbol type.

property enumerations

Returns an iterator of the Enumeration names.

classmethod file_symbol_url(*sub_path, filename=None*)

Returns an iterator of appropriate file-scheme symbol URLs that can be opened by a ResourceAccessor class.

Filter reduces the number of results returned to only those URLs containing that string

Return type*Generator[str, None, None]***get_enumeration**(**args, **kwargs*)**classmethod get_requirements**()

Returns a list of RequirementInterface objects required by this object.

Return type*List[RequirementInterface]***get_symbol**(**args, **kwargs*)

Resolves a symbol name into a symbol object.

If the symbol isn't found, it raises a SymbolError exception

get_symbol_type(*name*)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type*Optional[Template]***get_symbols_by_location**(*offset, size=0*)

Returns the name of all symbols in this table that live at a particular offset.

Return type*Iterable[str]*

get_symbols_by_type(*type_name*)

Returns the name of all symbols in this table that have type matching *type_name*.

Return type

`Iterable[str]`

get_type(*args, **kwargs)

Resolves a symbol name into an object template.

If the symbol isn't found it raises a `SymbolError` exception

get_type_class(*args, **kwargs)

Returns the class associated with a Symbol type.

classmethod make_subconfig(*context*, *base_config_path*, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property metadata

property natives: `NativeTableInterface`

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name*, *clazz*)

Calls the `set_type_class` function but does not throw an exception. Returns whether setting the type class was successful. :type name: `str` :param name: The name of the type to override the class for :type clazz: `Type[ObjectInterface]` :param clazz: The actual class to override for the provided type name

Return type

`bool`

provides = {'type': 'interface'}

set_type_class(*args, **kwargs)

Overrides the object class for a specific Symbol type.

Name *must* be present in self.types

Parameters

- **name** – The name of the type to override the class for
- **clazz** – The actual class to override for the provided type name

property symbols

Returns an iterator of the Symbol names.

property types

Returns an iterator of the Symbol type names.

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

class `MacUtilities(*args, **kwargs)`

Bases: `VersionableInterface`

Class with multiple useful mac functions.

classmethod `files_descriptors_for_process(context, symbol_table_name, task)`

Creates a generator for the file descriptors of a process

Parameters

- **symbol_table_name** (`str`) – The name of the symbol table associated with the process
- **context** (`ContextInterface`) –
- **task** (`ObjectInterface`) – The process structure to enumerate file descriptors from

Returns

- 1) The file's object
- 2) **The path referenced by the descriptor.**
The path is either empty, the full path of the file in the file system, or the formatted name for sockets, pipes, etc.
- 3) The file descriptor number

Return type

A 3 element tuple is yielded for each file descriptor

classmethod `generate_kernel_handler_info(context, layer_name, kernel, mods_list)`

classmethod `lookup_module_address(context, handlers, target_address, kernel_module_name=None)`

classmethod `mask_mods_list(context, layer_name, mods)`

A helper function to mask the starting and end address of kernel modules

Return type

`List[Tuple[ObjectInterface, Any, Any]]`

version = (1, 3, 0)

classmethod `walk_list_head(queue, next_member, max_elements=4096)`

Return type

`Iterable[ObjectInterface]`


```
classmethod walk_slist(queue, next_member, max_elements=4096)
```

Return type

`Iterable[ObjectInterface]`

```
classmethod walk_tailq(queue, next_member, max_elements=4096)
```

Return type

`Iterable[ObjectInterface]`

Subpackages

volatility3.framework.symbols.mac.extensions package

```
class fileglob(context, type_name, object_info, size, members)
```

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

```
class VolTemplateProxy
```

Bases: `VolTemplateProxy`

```
classmethod child_template(template, child)
```

Returns the template of a child to its parent.

Return type

`Template`

```
classmethod children(template)
```

Method to list children of a template.

Return type

`List[Template]`

```
classmethod has_member(template, member_name)
```

Returns whether the object would contain a member called member_name.

Return type

`bool`

```
classmethod relative_child_offset(template, child)
```

Returns the relative offset of a child to its parent.

Return type

`int`

```
classmethod replace_child(template, old_child, new_child)
```

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_fg_type()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

`object`

property `vol: ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class ifnet(*context, type_name, object_info, size, members*)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(attr='member')

Specifically named method for retrieving members.

Return type

`object`

sockaddr_dl()**property vol: *ReadOnlyMapping***

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

class inpcb(context, type_name, object_info, size, members)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object

- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_ipv4_info()

get_ipv6_info()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

str

get_tcp_state()

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class kauth_scope(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: `VolTemplateProxy`

classmethod child_template(*template*, *child*)

Returns the template of a child to its parent.

Return type

`Template`

classmethod children(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod has_member(*template*, *member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

classmethod relative_child_offset(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod replace_child(*template*, *old_child*, *new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod size(*template*)

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_listeners()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the *table_name* is not valid within the object's context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type`bool`**has_valid_members**(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**member**(*attr*='member')

Specifically named method for retrieving members.

Return type`object`**property vol:** `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class proc(*context, type_name, object_info, size, members*)

Bases: `GenericIntelProcess`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: `VolTemplateProxy`

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type`Template`**classmethod children**(*template*)

Method to list children of a template.

Return type`List[Template]`**classmethod has_member**(*template, member_name*)

Returns whether the object would contain a member called *member_name*.

Return type`bool`**classmethod relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

add_process_layer(*config_prefix=None, preferred_name=None*)

Constructs a new layer based on the process's DTB.

Returns the name of the Layer or None.

Return type

`Optional[str]`

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_map_iter()

Return type

`Iterable[ObjectInterface]`

get_process_memory_sections(*context, config_prefix, rw_no_file=False*)

Returns a list of sections based on the memory manager's view of this task's virtual memory.

Return type

`Generator[Tuple[int, int], None, None]`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

get_task()

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class queue_entry(*context, type_name, object_info, size, members*)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

bool

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

property *vol*: `ReadOnlyMapping`

Returns the volatility specific object information.

walk_list(*list_head*, *member_name*, *type_name*, *max_size*=4096)

Walks a queue in a smear-aware and smear-resistant manner

smear is detected by:

- the *max_size* parameter sets an upper bound
- each seen entry is only allowed once

attempts to work around smear:

- the list is walked in both directions to help find as many elements as possible

Parameters

- **list** (*type_name* - the type of each element in the) -
- **member** (*member_name* - the name of the embedded list) -
- **list** -
- **returned** (*max_size* - the maximum amount of elements that will be) -

Return type

`Iterable[ObjectInterface]`

Returns

Each instance of the queue cast as “*type_name*” type

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class `sockaddr`(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) - The context associated with the object
- **type_name** (`str`) - The name of the type structure for the object
- **object_info** (`ObjectInformation`) - Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`

Bases: `VolTemplateProxy`

classmethod `child_template`(*template*, *child*)

Returns the template of a child to its parent.

Return type

`Template`

classmethod children(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod has_member(*template*, *member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

classmethod relative_child_offset(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod replace_child(*template*, *old_child*, *new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod size(*template*)

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_address()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the *table_name* is not valid within the object's context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type`bool`**has_valid_members**(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**member**(*attr*='member')

Specifically named method for retrieving members.

Return type`object`**property vol:** `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class sockaddr_dl(*context, type_name, object_info, size, members*)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: `VolTemplateProxy`

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type`Template`**classmethod children**(*template*)

Method to list children of a template.

Return type`List[Template]`**classmethod has_member**(*template, member_name*)

Returns whether the object would contain a member called *member_name*.

Return type`bool`**classmethod relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

`object`

property vol: [*ReadOnlyMapping*](#)

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class socket(*context, type_name, object_info, size, members*)

Bases: [*StructType*](#)

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** ([*ContextInterface*](#)) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** ([*ObjectInformation*](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: [*VolTemplateProxy*](#)

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

[*Template*](#)

classmethod children(*template*)

Method to list children of a template.

Return type

[*List\[Template\]*](#)

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

[*bool*](#)

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

[*int*](#)

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

[*None*](#)

classmethod size(*template*)

Method to return the size of this type.

Return type

[*int*](#)

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

[*ObjectInterface*](#)

Note: If new type name does not include a symbol table, the symbol table for the current object is used

`get_connection_info()`

`get_converted_connection_info()`

`get_family()`

`get_inpcb()`

`get_protocol_as_string()`

`get_state()`

`get_symbol_table_name()`

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

`has_member(member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

`has_valid_member(member_name)`

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

`has_valid_members(member_names)`

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

`member(attr='member')`

Specifically named method for retrieving members.

Return type

`object`

property vol: `ReadOnlyMapping`

Returns the volatility specific object information.

`write(value)`

Writes the new value into the format at the offset the object currently resides at.

class `sysctl_oid`(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`

Bases: `VolTemplateProxy`

classmethod `child_template`(*template*, *child*)

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children`(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member`(*template*, *member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset`(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child`(*template*, *old_child*, *new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size`(*template*)

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_ctltype()

Returns the type of the sysctl node

Args: None

Returns

CTLTYPE_NODE CTLTYPE_INT CTLTYPE_STRING CTLTYPE_QUAD CTL-
TYPE_OPAQUE an empty string for nodes not in the above types

Return type

One of

Based on `sysctl_sysctl_debug_dump_node`

get_perms()

Returns the actions allowed on the node

Args: None

Returns

R - readable W - writeable L - self handles locking

Return type

A combination of

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(attr='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: [*ReadOnlyMapping*](#)

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class vm_map_entry(*context, type_name, object_info, size, members*)

Bases: [*StructType*](#)

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** ([*ContextInterface*](#)) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** ([*ObjectInformation*](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: [*VolTemplateProxy*](#)

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

[*Template*](#)

classmethod children(*template*)

Method to list children of a template.

Return type

[*List\[Template\]*](#)

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

[*bool*](#)

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

[*int*](#)

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

[*None*](#)

classmethod size(*template*)

Method to return the size of this type.

Return type

[*int*](#)

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

[*ObjectInterface*](#)

Note: If new type name does not include a symbol table, the symbol table for the current object is used

`get_object()`

`get_offset()`

`get_path(context, config_prefix)`

`get_perms()`

`get_range_alias()`

`get_special_path()`

`get_symbol_table_name()`

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

`get_vnode(context, config_prefix)`

`has_member(member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

`has_valid_member(member_name)`

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

`has_valid_members(member_names)`

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

`is_suspicious(context, config_prefix)`

Flags memory regions that are mapped rwx or that map an executable not back from a file on disk.

`member(attr='member')`

Specifically named method for retrieving members.

Return type

`object`

property vol: `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class **vm_map_object**(*context, type_name, object_info, size, members*)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class **VolTemplateProxy**

Bases: *VolTemplateProxy*

classmethod **child_template**(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod **children**(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod **has_member**(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod **relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod **replace_child**(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod **size**(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_map_object()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(attr='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

class vnode(context, type_name, object_info, size, members)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

ObjectInterface

Note: If new type name does not include a symbol table, the symbol table for the current object is used

full_path()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

str

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

volatility3.framework.symbols.windows package

class `WindowsKernelIntermedSymbols(*args, **kwargs)`

Bases: *IntermediateSymbolTable*

Instantiates a SymbolTable based on an IntermediateSymbolFormat JSON file. This is validated against the appropriate schema. The validation can be disabled by passing validate = False, but this should almost never be done.

Parameters

- **context** – The volatility context for the symbol table
- **config_path** – The configuration path for the symbol table
- **name** – The name for the symbol table (this is used in symbols e.g. table!symbol)
- **isf_url** – The URL pointing to the ISF file location
- **native_types** – The NativeSymbolTable that contains the native types for this symbol table
- **table_mapping** – A dictionary linking names referenced in the file with symbol tables in the context
- **validate** – Determines whether the ISF file will be validated against the appropriate schema
- **class_types** – A dictionary of type names and classes that override StructType when they are instantiated
- **symbol_mask** – An address mask used for all returned symbol offsets from this table (a mask of 0 disables masking)

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

clear_symbol_cache(*args, **kwargs)

Clears the symbol cache of this symbol table.

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod create(context, config_path, sub_path, filename, native_types=None, table_mapping=None, class_types=None, symbol_mask=0)

Takes a context and loads an intermediate symbol table based on a filename.

Parameters

- **context** (*ContextInterface*) – The context that the current plugin is being run within
- **config_path** (*str*) – The configuration path for reading/storing configuration information this symbol table may use
- **sub_path** (*str*) – The path under a suitable symbol path (defaults to volatility3/symbols and volatility3/framework/symbols) to check
- **filename** (*str*) – Basename of the file to find under the sub_path
- **native_types** (*Optional[NativeTableInterface]*) – Set of native types, defaults to native types read from the intermediate symbol format file
- **table_mapping** (*Optional[Dict[str, str]]*) – a dictionary of table names mentioned within the ISF file, and the tables within the context which they map to
- **symbol_mask** (*int*) – An address mask used for all returned symbol offsets from this table (a mask of 0 disables masking)

Return type

str

Returns

the name of the added symbol table

del_type_class(*args, **kwargs)

Removes the associated class override for a specific Symbol type.

property enumerations

Returns an iterator of the Enumeration names.

classmethod `file_symbol_url(sub_path, filename=None)`

Returns an iterator of appropriate file-scheme symbol URLs that can be opened by a ResourceAccessor class.

Filter reduces the number of results returned to only those URLs containing that string

Return type

`Generator[str, None, None]`

get_enumeration(*args, **kwargs)

classmethod `get_requirements()`

Returns a list of RequirementInterface objects required by this object.

Return type

`List[RequirementInterface]`

get_symbol(*args, **kwargs)

Resolves a symbol name into a symbol object.

If the symbol isn't found, it raises a SymbolError exception

get_symbol_type(name)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type

`Optional[Template]`

get_symbols_by_location(offset, size=0)

Returns the name of all symbols in this table that live at a particular offset.

Return type

`Iterable[str]`

get_symbols_by_type(type_name)

Returns the name of all symbols in this table that have type matching type_name.

Return type

`Iterable[str]`

get_type(*args, **kwargs)

Resolves a symbol name into an object template.

If the symbol isn't found it raises a SymbolError exception

get_type_class(*args, **kwargs)

Returns the class associated with a Symbol type.

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property metadata****property natives:** `NativeTableInterface`

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name*, *clazz*)

Calls the `set_type_class` function but does not throw an exception. Returns whether setting the type class was successful. :type name: `str` :param name: The name of the type to override the class for :type clazz: `Type[ObjectInterface]` :param clazz: The actual class to override for the provided type name

Return type`bool`**set_type_class**(*args, **kwargs)

Overrides the object class for a specific Symbol type.

Name *must* be present in `self.types`

Parameters

- **name** – The name of the type to override the class for
- **clazz** – The actual class to override for the provided type name

property symbols

Returns an iterator of the Symbol names.

property types

Returns an iterator of the Symbol type names.

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`**Subpackages****volatility3.framework.symbols.windows.extensions package****class CONTROL_AREA**(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `StructType`

A class for `_CONTROL_AREA` structures

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object

- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

PAGE_MASK = 4095

PAGE_SIZE = 4096

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod **child_template**(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod **children**(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod **has_member**(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod **relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod **replace_child**(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod **size**(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_available_pages()

Get the available pages that correspond to a cached file.

The tuples generated are (physical_offset, file_offset, page_size).

Return type

Iterable[Tuple[int, int, int]]

get_pte(*offset*)

Get a PTE object at the requested offset

Return type*ObjectInterface***get_subsection**()

Get the Subsection object, which is found immediately after the `_CONTROL_AREA`.

Return type*ObjectInterface***get_symbol_table_name**()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the `table_name` is not valid within the object's context

Return type*str***has_member**(*member_name*)

Returns whether the object would contain a member called `member_name`.

Return type*bool***has_valid_member**(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type*bool***has_valid_members**(*member_names*)

Returns whether the object has all of the members listed in `member_names`

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type*bool***is_valid**()

Determine if the object is valid.

Return type*bool***member**(*attr='member'*)

Specifically named method for retrieving members.

Return type*object*

property vol: [*ReadOnlyMapping*](#)

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class `DEVICE_OBJECT`(*context, type_name, object_info, size, members*)

Bases: [*StructType*](#), [*ExecutiveObject*](#)

A class for kernel device objects.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** ([*ContextInterface*](#)) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** ([*ObjectInformation*](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`

Bases: [*VolTemplateProxy*](#)

classmethod `child_template`(*template, child*)

Returns the template of a child to its parent.

Return type

[*Template*](#)

classmethod `children`(*template*)

Method to list children of a template.

Return type

[*List*](#)[[*Template*](#)]

classmethod `has_member`(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod `relative_child_offset`(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod `replace_child`(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod `size`(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

[*ObjectInterface*](#)

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_attached_devices()

Enumerate the attached device's objects

Return type

`Generator[ObjectInterface, None, None]`

get_device_name()

Get device's name from the object header.

Return type

`str`

get_object_header()**Return type**

`OBJECT_HEADER`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(attr='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: [ReadOnlyMapping](#)

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class DRIVER_OBJECT(*context, type_name, object_info, size, members*)

Bases: [StructType](#), [ExecutiveObject](#)

A class for kernel driver objects.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** ([ContextInterface](#)) – The context associated with the object
- **type_name** ([str](#)) – The name of the type structure for the object
- **object_info** ([ObjectInformation](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: [VolTemplateProxy](#)

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

[Template](#)

classmethod children(*template*)

Method to list children of a template.

Return type

[List](#)[[Template](#)]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

[bool](#)

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

[int](#)

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

[None](#)

classmethod size(*template*)

Method to return the size of this type.

Return type

[int](#)

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

[ObjectInterface](#)

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_devices()

Enumerate the driver's device objects

Return type

`Generator[ObjectInterface, None, None]`

get_driver_name()

Get driver's name from the object header.

Return type

`str`

get_object_header()**Return type**

`OBJECT_HEADER`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

is_valid()

Determine if the object is valid.

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class `EPROCESS`(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `GenericIntelProcess`, `ExecutiveObject`

A class for executive kernel processes objects.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`

Bases: `VolTemplateProxy`

classmethod `child_template`(*template*, *child*)

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children`(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member`(*template*, *member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset`(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child`(*template*, *old_child*, *new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size`(*template*)

Method to return the size of this type.

Return type

`int`

add_process_layer(*config_prefix=None, preferred_name=None*)

Constructs a new layer based on the process's DirectoryTableBase.

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

environment_variables()

Generator for environment variables.

The PEB points to our env block - a series of null-terminated unicode strings. Each string cannot be more than 0x7FFF chars. End of the list is a quad-null.

get_create_time()

get_exit_time()

get_handle_count()

get_is_wow64()

get_object_header()

Return type

OBJECT_HEADER

get_peb()

Constructs a PEB object

Return type

ObjectInterface

get_session_id()

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

str

get_vad_root()

get_wow_64_process()

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

init_order_modules()

Generator for DLLs in the order that they were initialized

Return type

Iterable[ObjectInterface]

is_valid()

Determine if the object is valid.

Return type

bool

load_order_modules()

Generator for DLLs in the order that they were loaded.

Return type

Iterable[ObjectInterface]

mem_order_modules()

Generator for DLLs in the order that they appear in memory

Return type

Iterable[ObjectInterface]

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class ETHREAD(*context, type_name, object_info, size, members*)

Bases: *StructType*

A class for executive thread objects.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_cross_thread_flags()**Return type**

str

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type`str`**has_member**(*member_name*)

Returns whether the object would contain a member called member_name.

Return type`bool`**has_valid_member**(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters**member_name** (`str`) – Name of the member to test access to determine if the member is valid or not**Return type**`bool`**has_valid_members**(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters**member_names** (`List[str]`) – List of names to test as to members with those names validity**Return type**`bool`**member**(*attr*='member')

Specifically named method for retrieving members.

Return type`object`**owning_process**()

Return the EPROCESS that owns this thread.

Return type`ObjectInterface`**property vol:** `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class EX_FAST_REF(*context*, *type_name*, *object_info*, *size*, *members*)Bases: `StructType`

This is a standard Windows structure that stores a pointer to an object but also leverages the least significant bits to encode additional details.

When dereferencing the pointer, we need to strip off the extra bits.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object

- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod **child_template**(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod **children**(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod **has_member**(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod **relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod **replace_child**(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod **size**(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

dereference()

Return type

ObjectInterface

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type`str`**has_member**(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type`bool`**has_valid_member**(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type`bool`**has_valid_members**(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**member**(*attr*='member')

Specifically named method for retrieving members.

Return type`object`**property** **vol**: `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class **FILE_OBJECT**(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `StructType`, `ExecutiveObject`

A class for windows file objects.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class **VolTemplateProxy**

Bases: `VolTemplateProxy`

classmethod **child_template**(*template*, *child*)

Returns the template of a child to its parent.

Return type*Template***classmethod children**(*template*)

Method to list children of a template.

Return type*List[Template]***classmethod has_member**(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type*bool***classmethod relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type*int***classmethod replace_child**(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type*None***classmethod size**(*template*)

Method to return the size of this type.

Return type*int***access_string**()**cast**(*new_type_name, **additional*)Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

file_name_with_device()**Return type***Union[str, BaseAbsentValue]***get_object_header**()**Return type***OBJECT_HEADER***get_symbol_table_name**()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type*str*

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

is_valid()

Determine if the object is valid.

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

property **vol**: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class **KMUTANT**(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: *StructType*, *ExecutiveObject*

A class for windows mutant objects.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class **VolTemplateProxy**

Bases: *VolTemplateProxy*

classmethod `child_template(template, child)`

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children(template)`

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member(template, member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_name()

Get the object's name from the object header.

Return type

`str`

get_object_header()

Return type

`OBJECT_HEADER`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

is_valid()

Determine if the object is valid.

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

property **vol**: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class **KSYSTEM_TIME**(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: *StructType*

A system time structure that stores a high and low part.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class **VolTemplateProxy**

Bases: *VolTemplateProxy*

classmethod `child_template(template, child)`

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children(template)`

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member(template, member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(new_type_name, **additional)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

get_time()

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class KTHREAD(*context, type_name, object_info, size, members*)

Bases: *StructType*

A class for thread control block objects.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod `has_member(template, member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_state()

Return type

`str`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

get_wait_reason()

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class LIST_ENTRY(*context, type_name, object_info, size, members*)

Bases: *StructType, Iterable*

A class for double-linked lists on Windows.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

bool

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

to_list(*symbol_type*, *member*, *forward*=True, *sentinel*=True, *layer*=None)

Returns an iterator of the entries in the list.

Return type

`Iterator[ObjectInterface]`

property vol: `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class MMVAD(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `MMVAD_SHORT`

A version of the process virtual memory range structure that contains additional fields necessary to map files from disk.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: `VolTemplateProxy`

classmethod child_template(*template*, *child*)

Returns the template of a child to its parent.

Return type

`Template`

classmethod children(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod has_member(*template*, *member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod relative_child_offset(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_commit_charge()

Get the VAD's commit charge (number of committed pages)

get_end()

Get the VAD's ending virtual address. This is the last accessible byte in the range.

Return type

`int`

get_file_name()

Get the name of the file mapped into the memory range (if any)

get_left_child()

Get the left child member.

get_parent()

Get the VAD's parent member.

get_private_memory()

Get the VAD's private memory setting.

get_protection(*protect_values*, *winnt_protections*)

Get the VAD's protection constants as a string.

get_right_child()

Get the right child member.

get_size()

Get the size of the VAD region. The OS ensures page granularity.

Return type

`int`

get_start()

Get the VAD's starting virtual address. This is the first accessible byte in the range.

Return type

`int`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

get_tag()**has_member(member_name)**

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(attr='member')

Specifically named method for retrieving members.

Return type

`object`

traverse(visited=None, depth=0)

Traverse the VAD tree, determining each underlying VAD node type by looking up the pool tag for the structure and then casting into a new object.

property vol: [ReadOnlyMapping](#)

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

class MMVAD_SHORT(context, type_name, object_info, size, members)

Bases: [StructType](#)

A class that represents process virtual memory ranges.

Each instance is a node in a binary tree structure and is pointed to by VadRoot.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_commit_charge()

Get the VAD's commit charge (number of committed pages)

get_end()

Get the VAD's ending virtual address. This is the last accessible byte in the range.

Return type

int

get_file_name()

Only long(er) vads have mapped files.

get_left_child()

Get the left child member.

get_parent()

Get the VAD's parent member.

get_private_memory()

Get the VAD's private memory setting.

get_protection(*protect_values*, *winnt_protections*)

Get the VAD's protection constants as a string.

get_right_child()

Get the right child member.

get_size()

Get the size of the VAD region. The OS ensures page granularity.

Return type

`int`

get_start()

Get the VAD's starting virtual address. This is the first accessible byte in the range.

Return type

`int`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

get_tag()**has_member(*member_name*)**

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

traverse(*visited=None, depth=0*)

Traverse the VAD tree, determining each underlying VAD node type by looking up the pool tag for the structure and then casting into a new object.

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class OBJECT_SYMBOLIC_LINK(*context, type_name, object_info, size, members*)

Bases: *StructType, ExecutiveObject*

A class for kernel link objects.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

bool

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

ObjectInterface

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_create_time()

get_link_name()

Return type

`str`

get_object_header()

Return type

OBJECT_HEADER

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type`bool`**has_valid_members**(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**is_valid**()

Determine if the object is valid.

Return type`bool`**member**(*attr='member'*)

Specifically named method for retrieving members.

Return type`object`**property vol:** *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class SHARED_CACHE_MAP(*context, type_name, object_info, size, members*)

Bases: *StructType*

A class for `_SHARED_CACHE_MAP` structures

Constructs an Object adhering to the `ObjectInterface`.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

VACB_ARRAY = 128**VACB_BLOCK** = 262144**VACB_LEVEL_SHIFT** = 7**VACB_OFFSET_SHIFT** = 18**VACB_SIZE_OF_FIRST_LEVEL** = 33554432**class VolTemplateProxy**

Bases: *VolTemplateProxy*

classmethod `child_template(template, child)`

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children(template)`

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member(template, member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_available_pages()

Get the available pages that correspond to a cached file.

The lists generated are (virtual_offset, file_offset, page_size).

Return type

`List`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

is_valid()

Determine if the object is valid.

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

process_index_array(*array_pointer*, *level*, *limit*, *vacb_list*=None)

Recursively process the sparse multilevel VACB index array.

Parameters

- **array_pointer** (`ObjectInterface`) – The address of a possible index array
- **level** (`int`) – The current level
- **limit** (`int`) – The level where we abandon all hope. Ideally this is 7
- **vacb_list** (`Optional[List]`) – An array of collected VACBs

Return type

`List`

Returns

Collected VACBs

save_vacb(*vacb_obj*, *vacb_list*)

property vol: `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class **TOKEN**(*context, type_name, object_info, size, members*)

Bases: *StructType*

A class for process etoken object.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class **VolTemplateProxy**

Bases: *VolTemplateProxy*

classmethod **child_template**(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod **children**(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod **has_member**(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod **relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod **replace_child**(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod **size**(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_sids()

Yield a sid for the current token object.

Return type

`Iterable[str]`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(attr='member')

Specifically named method for retrieving members.

Return type

`object`

privileges()

Return a list of privileges for the current token object.

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

class UNICODE_STRING(*context, type_name, object_info, size, members*)

Bases: [StructType](#)

A class for Windows unicode string structures.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** ([ContextInterface](#)) – The context associated with the object
- **type_name** ([str](#)) – The name of the type structure for the object
- **object_info** ([ObjectInformation](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

property String: [ObjectInterface](#)

class VolTemplateProxy

Bases: [VolTemplateProxy](#)

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

[Template](#)

classmethod children(*template*)

Method to list children of a template.

Return type

[List\[Template\]](#)

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

[bool](#)

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

[int](#)

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

[None](#)

classmethod size(*template*)

Method to return the size of this type.

Return type

[int](#)

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

[ObjectInterface](#)

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_string()

Return type

ObjectInterface

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

str

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

bool

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(attr='member')

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

class VACB(context, type_name, object_info, size, members)

Bases: *StructType*

A class for _VACB structures

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

FILEOFFSET_MASK = 18446744073709486080

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

ObjectInterface

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_file_offset()

Return type

int

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

str

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (str) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (List[str]) – List of names to test as to members with those names validity

Return type

bool

member(*attr*='member')

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

Submodules

volatility3.framework.symbols.windows.extensions.crash module

class **SUMMARY_DUMP**(*context, type_name, object_info, size, members*)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (str) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxyBases: *VolTemplateProxy***classmethod child_template**(*template, child*)

Returns the template of a child to its parent.

Return type*Template***classmethod children**(*template*)

Method to list children of a template.

Return type*List[Template]***classmethod has_member**(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type*bool***classmethod relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type*int***classmethod replace_child**(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type*None***classmethod size**(*template*)

Method to return the size of this type.

Return type*int***cast**(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

ObjectInterface

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_buffer(*sub_type, count*)**Return type***ObjectInterface***get_buffer_char**()**Return type***ObjectInterface***get_buffer_long**()**Return type***ObjectInterface***get_symbol_table_name**()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

volatility3.framework.symbols.windows.extensions.kdbg module

class **KDDEBUGGER_DATA64**(*context, type_name, object_info, size, members*)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxyBases: *VolTemplateProxy***classmethod child_template**(*template, child*)

Returns the template of a child to its parent.

Return type*Template***classmethod children**(*template*)

Method to list children of a template.

Return type*List[Template]***classmethod has_member**(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type*bool***classmethod relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type*int***classmethod replace_child**(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type*None***classmethod size**(*template*)

Method to return the size of this type.

Return type*int***cast**(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

ObjectInterface

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_build_lab()

Returns the NT build lab string from the KDBG.

get_csdversion()

Returns the CSDVersion as an integer (i.e. Service Pack number)

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type*str*

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: [*ReadOnlyMapping*](#)

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

volatility3.framework.symbols.windows.extensions.mbr module

class `PARTITION_ENTRY`(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: [*StructType*](#)

Constructs an Object adhering to the `ObjectInterface`.

Parameters

- **context** ([*ContextInterface*](#)) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** ([*ObjectInformation*](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`

Bases: [*VolTemplateProxy*](#)

classmethod `child_template`(*template*, *child*)

Returns the template of a child to its parent.

Return type

[*Template*](#)

classmethod children(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod has_member(*template*, *member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

classmethod relative_child_offset(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod replace_child(*template*, *old_child*, *new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod size(*template*)

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_bootable_flag()

Get Bootable Flag.

Return type

`int`

get_ending_chs()

Get Ending CHS (Cylinder Header Sector) Address.

Return type

`int`

get_ending_cylinder()

Get Ending Cylinder.

Return type

`int`

get_ending_sector()

Get Ending Sector.

Return type

`int`

get_partition_type()

Get Partition Type.

Return type

`str`

get_size_in_sectors()

Get Size in Sectors.

Return type

`int`

get_starting_chs()

Get Starting CHS (Cylinder Header Sector) Address.

Return type

`int`

get_starting_cylinder()

Get Starting Cylinder.

Return type

`int`

get_starting_lba()

Get Starting LBA (Logical Block Addressing).

Return type

`int`

get_starting_sector()

Get Starting Sector.

Return type

`int`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

is_bootable()

Check Bootable Partition.

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class PARTITION_TABLE(*context, type_name, object_info, size, members*)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

bool

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_disk_signature()

Get Disk Signature (GUID).

Return type

`str`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

volatility3.framework.symbols.windows.extensions.mft module

class MFTAttribute(*context, type_name, object_info, size, members*)

Bases: `StructType`

This represents an MFT ATTRIBUTE

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: `VolTemplateProxy`

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

`Template`

classmethod children(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_resident_filecontent()

Return type

`bytes`

get_resident_filename()

Return type

`str`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**member**(*attr*='member')

Specifically named method for retrieving members.

Return type`object`**property** **vol:** `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class **MFTEntry**(*context*, *type_name*, *object_info*, *size*, *members*)Bases: `StructType`

This represents the base MFT Record

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class **VolTemplateProxy**Bases: `VolTemplateProxy`**classmethod** **child_template**(*template*, *child*)

Returns the template of a child to its parent.

Return type`Template`**classmethod** **children**(*template*)

Method to list children of a template.

Return type`List[Template]`**classmethod** **has_member**(*template*, *member_name*)

Returns whether the object would contain a member called member_name.

Return type`bool`**classmethod** **relative_child_offset**(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type`int`**classmethod** **replace_child**(*template*, *old_child*, *new_child*)

Replace a child elements within the arguments handed to the template.

Return type`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_signature()

Return type

`str`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

`object`

property vol: [*ReadOnlyMapping*](#)

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class MFTFileName(*context, type_name, object_info, size, members*)

Bases: [*StructType*](#)

This represents an MFT \$FILE_NAME Attribute

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** ([*ContextInterface*](#)) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** ([*ObjectInformation*](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: [*VolTemplateProxy*](#)

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

[*Template*](#)

classmethod children(*template*)

Method to list children of a template.

Return type

[*List\[Template\]*](#)

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

[*ObjectInterface*](#)

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_full_name()

Return type

`str`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

member(attr='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

volatility3.framework.symbols.windows.extensions.network module

inet_ntop(*address_family*, *packed_ip*)

Return type
`str`

volatility3.framework.symbols.windows.extensions.pe module

class IMAGE_DOS_HEADER(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `StructType`

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: `VolTemplateProxy`

classmethod child_template(*template*, *child*)

Returns the template of a child to its parent.

Return type
`Template`

classmethod children(*template*)

Method to list children of a template.

Return type
`List[Template]`

classmethod has_member(*template*, *member_name*)

Returns whether the object would contain a member called member_name.

Return type
`bool`

classmethod relative_child_offset(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type
`int`

classmethod replace_child(*template*, *old_child*, *new_child*)

Replace a child elements within the arguments handed to the template.

Return type
`None`

classmethod size(*template*)

Method to return the size of this type.

Return type
`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

fix_image_base(*raw_data*, *nt_header*)

Fix the _OPTIONAL_HEADER.ImageBase value (which is either an unsigned long for 32-bit PE's or unsigned long long for 64-bit PE's) to match the address where the PE file was carved out of memory.

Parameters

- **raw_data** (*bytes*) – a bytes object of the PE's data
- **nt_header** (*ObjectInterface*) – <_IMAGE_NT_HEADERS> or <_IMAGE_NT_HEADERS64> instance

Return type

bytes

Returns

<bytes> patched with the correct address

get_nt_header()

Carve out the NT header from this DOS header. This reflects on the PE file's Machine type to create a 32- or 64-bit NT header structure.

Return type

ObjectInterface

Returns

<_IMAGE_NT_HEADERS> or <_IMAGE_NT_HEADERS64> instance

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

str

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (*List*[*str*]) – List of names to test as to members with those names validity

Return type

bool

member(*attr*='member')

Specifically named method for retrieving members.

Return type

object

reconstruct()

This method generates the content necessary to reconstruct a PE file from memory. It preserves slack space (similar to the old –memory) and automatically fixes the ImageBase in the output PE file.

Return type

Generator[*Tuple*[*int*, *bytes*], *None*, *None*]

Returns

<tuple> of (<int> offset, <bytes> data)

replace_header_field(*sect*, *header*, *item*, *value*)

Replaces a member in an _IMAGE_SECTION_HEADER structure.

Parameters

- **sect** (*ObjectInterface*) – the section instance
- **header** (*bytes*) – raw data for the section
- **item** (*ObjectInterface*) – the member of the section to replace
- **value** (*int*) – new value for the member

Return type

bytes

Returns

The raw data with the replaced header field

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class IMAGE_NT_HEADERS(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

ObjectInterface

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_sections()

Iterate through the section headers for this PE file.

Yields

<_IMAGE_SECTION_HEADER> objects

Return type

Generator[ObjectInterface, None, None]

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type`str`**has_member**(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type`bool`**has_valid_member**(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type`bool`**has_valid_members**(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**member**(*attr*='member')

Specifically named method for retrieving members.

Return type`object`**property** **vol:** `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

volatility3.framework.symbols.windows.extensions.pool module**class** **ExecutiveObject**(*context*, *type_name*, *object_info*, ***kwargs*)

Bases: `ObjectInterface`

This is used as a “mixin” that provides all kernel executive objects with a means of finding their own object header.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxyBases: `object`

A container for proxied methods that the ObjectTemplate of this object will call. This is primarily to keep methods together for easy organization/management, there is no significant need for it to be a separate class.

The methods of this class *must* be class methods rather than standard methods, to allow for code reuse. Each method also takes a template since the templates may contain the necessary data about the yet-to-be-constructed object. It allows objects to control how their templates respond without needing to write new templates for each and every potential object type.

abstract classmethod child_template(*template, child*)

Returns the template of the child member from the parent.

Return type`Template`**abstract classmethod children**(*template*)

Returns the children of the template.

Return type`List[Template]`**abstract classmethod has_member**(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type`bool`**abstract classmethod relative_child_offset**(*template, child*)

Returns the relative offset from the head of the parent data to the child member.

Return type`int`**abstract classmethod replace_child**(*template, old_child, new_child*)

Substitutes the old_child for the new_child.

Return type`None`**abstract classmethod size**(*template*)

Returns the size of the template object.

Return type`int`**cast**(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_object_header()**Return type**`OBJECT_HEADER`**get_symbol_table_name**()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type`str`**has_member**(*member_name*)

Returns whether the object would contain a member called member_name.

Parameters

member_name (`str`) – Name to test whether a member exists within the type structure

Return type`bool`**has_valid_member**(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type`bool`**has_valid_members**(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**property vol:** *ReadOnlyMapping*

Returns the volatility specific object information.

abstract write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class OBJECT_HEADER(*context, type_name, object_info, size, members*)

Bases: *StructType*

A class for the headers for executive kernel objects, which contains quota information, ownership details, naming data, and ACLs.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

property NameInfo: *ObjectInterface***class VolTemplateProxy**

Bases: *VolTemplateProxy*

classmethod `child_template(template, child)`

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children(template)`

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member(template, member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_object_type(*type_map*, *cookie=None*)

Across all Windows versions, the _OBJECT_HEADER embeds details on the type of object (i.e. process, file) but the way its embedded differs between versions.

This API abstracts away those details.

Return type

`Optional[str]`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

is_valid()

Determine if the object is valid.

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

property **vol**: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class **POOL_HEADER**(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: *StructType*

A kernel pool allocation header.

Exists at the base of the allocation and provides a tag that we can scan for.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod size(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

ObjectInterface

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_object(*constraint, use_top_down, kernel_symbol_table=None, native_layer_name=None*)

Carve an object or data structure from a kernel pool allocation

Parameters

- **constraint** (*PoolConstraint*) – a PoolConstraint object used to get the pool allocation header object
- **use_top_down** (*bool*) – for delineating how a windows version finds the size of the object body
- **kernel_symbol_table** (*Optional[str]*) – in case objects of a different symbol table are scanned for
- **native_layer_name** (*Optional[str]*) – the name of the layer where the data originally lived

Return type

Optional[ObjectInterface]

Returns

An object as found from a POOL_HEADER

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(member_name)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(member_name)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(member_names)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

is_free_pool()**is_nonpaged_pool()****is_paged_pool()****member(attr='member')**

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(value)

Writes the new value into the format at the offset the object currently resides at.

class `POOL_HEADER_VISTA`(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `POOL_HEADER`

A kernel pool allocation header, updated for Vista and later.

Exists at the base of the allocation and provides a tag that we can scan for.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class `VolTemplateProxy`

Bases: `VolTemplateProxy`

classmethod `child_template`(*template*, *child*)

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children`(*template*)

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member`(*template*, *member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset`(*template*, *child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child`(*template*, *old_child*, *new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size`(*template*)

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_object(*constraint*, *use_top_down*, *kernel_symbol_table=None*, *native_layer_name=None*)

Carve an object or data structure from a kernel pool allocation

Parameters

- **constraint** (*PoolConstraint*) – a PoolConstraint object used to get the pool allocation header object
- **use_top_down** (*bool*) – for delineating how a windows version finds the size of the object body
- **kernel_symbol_table** (*Optional[str]*) – in case objects of a different symbol table are scanned for
- **native_layer_name** (*Optional[str]*) – the name of the layer where the data originally lived

Return type

Optional[ObjectInterface]

Returns

An object as found from a POOL_HEADER

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

str

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in member_names

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

is_free_pool()

is_nonpaged_pool()

is_paged_pool()

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: [*ReadOnlyMapping*](#)

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class POOL_TRACKER_BIG_PAGES(*context, type_name, object_info, size, members*)

Bases: [*StructType*](#)

A kernel big page pool tracker.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** ([*ContextInterface*](#)) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** ([*ObjectInformation*](#)) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: [*VolTemplateProxy*](#)

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

[*Template*](#)

classmethod children(*template*)

Method to list children of a template.

Return type

[*List\[Template\]*](#)

classmethod has_member(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod relative_child_offset(*template, child*)

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod replace_child(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_key()

Returns the Key value as a 4 character string

Return type

`str`

get_number_of_bytes()

Returns the NumberOfBytes value on applicable systems

Return type

`Union[int, BaseAbsentValue]`

get_pool_type()

Returns the enum name for the PoolType value on applicable systems

Return type

`Union[str, BaseAbsentValue]`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

is_free()

Returns if the allocation is freed (True) or in-use (False)

Return type

bool

is_valid()

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

pool_type_lookup: *Dict[str, str]* = {}

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

volatility3.framework.symbols.windows.extensions.registry module

class CMHIVE(*context, type_name, object_info, size, members*)

Bases: *StructType*

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod `children(template)`

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member(template, member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_name()

Determine a name for the hive.

Note that some attributes are unpredictably blank across different OS versions while others are populated, so we check all possibilities and take the first one that's not empty

Return type

`Optional[ObjectInterface]`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

is_valid()

Determine if the object is valid.

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property name: *ObjectInterface* | *None*

Determine a name for the hive.

Note that some attributes are unpredictably blank across different OS versions while others are populated, so we check all possibilities and take the first one that's not empty

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class CM_KEY_BODY(*context, type_name, object_info, size, members*)

Bases: *StructType*

This represents an open handle to a registry key and is not tied to the registry hive file format on disk.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod `child_template(template, child)`

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children(template)`

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member(template, member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_full_key_name()

Return type

`str`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object’s symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object’s context

Return type

`str`

has_member(*member_name*)

Returns whether the object would contain a member called member_name.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class CM_KEY_NODE(*context, type_name, object_info, size, members*)

Bases: *StructType*

Extension to allow traversal of registry keys.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod child_template(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod children(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod `has_member(template, member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: `ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_key_path()

Return type

`str`

get_name()

Gets the name for the current key node

Return type

`ObjectInterface`

get_subkeys()

Returns a list of the key nodes.

Return type

`Iterable[ObjectInterface]`

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

`str`

get_values()

Returns a list of the Value nodes for a key.

Return type`Iterable[ObjectInterface]`**get_volatile()****Return type**`bool`**has_member**(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type`bool`**has_valid_member**(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type`bool`**has_valid_members**(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**member**(*attr*='member')

Specifically named method for retrieving members.

Return type`object`**property** **vol:** `ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class **CM_KEY_VALUE**(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `StructType`

Extensions to extract data from CM_KEY_VALUE nodes.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxyBases: *VolTemplateProxy***classmethod child_template**(*template, child*)

Returns the template of a child to its parent.

Return type*Template***classmethod children**(*template*)

Method to list children of a template.

Return type*List[Template]***classmethod has_member**(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type*bool***classmethod relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type*int***classmethod replace_child**(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type*None***classmethod size**(*template*)

Method to return the size of this type.

Return type*int***cast**(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

ObjectInterface

Note: If new type name does not include a symbol table, the symbol table for the current object is used

decode_data()

Properly decodes the data associated with the value node

Return type*Union[int, bytes]***get_name**()

Gets the name for the current key value

Return type*ObjectInterface***get_symbol_table_name**()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table

- **KeyError** – If the `table_name` is not valid within the object's context

Return type`str`**has_member**(*member_name*)

Returns whether the object would contain a member called `member_name`.

Return type`bool`**has_valid_member**(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type`bool`**has_valid_members**(*member_names*)

Returns whether the object has all of the members listed in `member_names`

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**member**(*attr*='member')

Specifically named method for retrieving members.

Return type`object`**property** `vol: ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class **HMAP_ENTRY**(*context*, *type_name*, *object_info*, *size*, *members*)

Bases: `StructType`

Constructs an Object adhering to the `ObjectInterface`.

Parameters

- **context** (`ContextInterface`) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (`ObjectInformation`) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class **VolTemplateProxy**

Bases: `VolTemplateProxy`

classmethod **child_template**(*template*, *child*)

Returns the template of a child to its parent.

Return type*Template***classmethod children**(*template*)

Method to list children of a template.

Return type*List[Template]***classmethod has_member**(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type*bool***classmethod relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type*int***classmethod replace_child**(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type*None***classmethod size**(*template*)

Method to return the size of this type.

Return type*int***cast**(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

ObjectInterface

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_block_offset()**Return type***int***get_symbol_table_name**()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type*str***has_member**(*member_name*)

Returns whether the object would contain a member called member_name.

Return type*bool*

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (*str*) – Name of the member to test access to determine if the member is valid or not

Return type

bool

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (*List[str]*) – List of names to test as to members with those names validity

Return type

bool

member(*attr='member'*)

Specifically named method for retrieving members.

Return type

object

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class RegKeyFlags(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: *IntEnum*

KEY_COMP_NAME = 32

KEY_HIVE_ENTRY = 4

KEY_HIVE_EXIT = 2

KEY_IS_VOLATILE = 1

KEY_NO_DELETE = 8

KEY_PREFEF_HANDLE = 64

KEY_SYM_LINK = 16

KEY_VIRTUAL_STORE = 512

KEY_VIRT_MIRRORED = 128

KEY_VIRT_TARGET = 256

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If `byteorder` is 'big', the most significant byte is at the beginning of the byte array. If `byteorder` is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If `signed` is `False` and a negative integer is given, an `OverflowError` is raised.

class RegValueTypes(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `Enum`

`REG_BINARY = 3`

`REG_DWORD = 4`

`REG_DWORD_BIG_ENDIAN = 5`

`REG_EXPAND_SZ = 2`

`REG_FULL_RESOURCE_DESCRIPTOR = 9`

`REG_LINK = 6`

`REG_MULTI_SZ = 7`

`REG_NONE = 0`

`REG_QWORD = 11`

`REG_RESOURCE_LIST = 8`

`REG_RESOURCE_REQUIREMENTS_LIST = 10`

`REG_SZ = 1`

`REG_UNKNOWN = 99999`

volatility3.framework.symbols.windows.extensions.services module

class SERVICE_HEADER(*context, type_name, object_info, size, members*)

Bases: `StructType`

A service header structure.

Constructs an Object adhering to the `ObjectInterface`.

Parameters

- **context** (`ContextInterface`) – The context associated with the object

- **type_name** (*str*) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod **child_template**(*template, child*)

Returns the template of a child to its parent.

Return type

Template

classmethod **children**(*template*)

Method to list children of a template.

Return type

List[Template]

classmethod **has_member**(*template, member_name*)

Returns whether the object would contain a member called member_name.

Return type

bool

classmethod **relative_child_offset**(*template, child*)

Returns the relative offset of a child to its parent.

Return type

int

classmethod **replace_child**(*template, old_child, new_child*)

Replace a child elements within the arguments handed to the template.

Return type

None

classmethod **size**(*template*)

Method to return the size of this type.

Return type

int

cast(*new_type_name, **additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype: *ObjectInterface*

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_symbol_table_name()

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type

str

has_member(*member_name*)

Returns whether the object would contain a member called *member_name*.

Return type

`bool`

has_valid_member(*member_name*)

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type

`bool`

has_valid_members(*member_names*)

Returns whether the object has all of the members listed in *member_names*

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type

`bool`

is_valid()

Determine if the structure is valid.

Return type

`bool`

member(*attr*='member')

Specifically named method for retrieving members.

Return type

`object`

property vol: *ReadOnlyMapping*

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

class SERVICE_RECORD(*context, type_name, object_info, size, members*)

Bases: *StructType*

A service record structure.

Constructs an Object adhering to the ObjectInterface.

Parameters

- **context** (*ContextInterface*) – The context associated with the object
- **type_name** (`str`) – The name of the type structure for the object
- **object_info** (*ObjectInformation*) – Basic information relevant to the object (layer, offset, member_name, parent, etc)

class VolTemplateProxy

Bases: *VolTemplateProxy*

classmethod `child_template(template, child)`

Returns the template of a child to its parent.

Return type

`Template`

classmethod `children(template)`

Method to list children of a template.

Return type

`List[Template]`

classmethod `has_member(template, member_name)`

Returns whether the object would contain a member called member_name.

Return type

`bool`

classmethod `relative_child_offset(template, child)`

Returns the relative offset of a child to its parent.

Return type

`int`

classmethod `replace_child(template, old_child, new_child)`

Replace a child elements within the arguments handed to the template.

Return type

`None`

classmethod `size(template)`

Method to return the size of this type.

Return type

`int`

cast(*new_type_name*, ***additional*)

Returns a new object at the offset and from the layer that the current object inhabits. :rtype:

`ObjectInterface`

Note: If new type name does not include a symbol table, the symbol table for the current object is used

get_binary()

Returns the binary associated with the service.

Return type

`Union[str, BaseAbsentValue]`

get_display()

Returns the service display.

Return type

`Union[str, BaseAbsentValue]`

get_name()

Returns the service name.

Return type

`Union[str, BaseAbsentValue]`

get_pid()

Return the pid of the process, if any.

Return type`Union[int, BaseAbsentValue]`**get_symbol_table_name()**

Returns the symbol table name for this particular object.

Raises

- **ValueError** – If the object's symbol does not contain an explicit table
- **KeyError** – If the table_name is not valid within the object's context

Return type`str`**get_type()**

Returns the binary types.

Return type`str`**has_member(member_name)**

Returns whether the object would contain a member called member_name.

Return type`bool`**has_valid_member(member_name)**

Returns whether the dereferenced type has a valid member.

Parameters

member_name (`str`) – Name of the member to test access to determine if the member is valid or not

Return type`bool`**has_valid_members(member_names)**

Returns whether the object has all of the members listed in member_names

Parameters

member_names (`List[str]`) – List of names to test as to members with those names validity

Return type`bool`**is_valid()**

Determine if the structure is valid.

Return type`bool`**member(attr='member')**

Specifically named method for retrieving members.

Return type`object`**traverse()**

Generator that enumerates other services.

property `vol: ReadOnlyMapping`

Returns the volatility specific object information.

write(*value*)

Writes the new value into the format at the offset the object currently resides at.

Submodules

`volatility3.framework.symbols.windows.pdbconv` module

class `ForwardArrayCount`(*size, element_type*)

Bases: `object`

class `PdbReader`(*context, location, database_name=None, progress_callback=None*)

Bases: `object`

Class to read Microsoft PDB files.

This reads the various streams according to various sources as to how pdb should be read. These sources include:

<https://docs.rs/crate/pdb/0.5.0/source/src/> <https://github.com/moyix/pdbparse> <https://llvm.org/docs/PDB/index.html> <https://github.com/Microsoft/microsoft-pdb/>

In order to generate ISF files, we need the type stream (2), and the symbols stream (variable). The MultiStream Format wrapper is handled as a volatility layer, which constructs sublayers for each stream. The streams can then be read contiguously allowing the data to be accessed.

Volatility's type system is strong when everything must be laid out in advance, but PDB data is reasonably dynamic, particularly when it comes to names. We must therefore parse it after we've collected other information already. This is in comparison to something such as Construct/pdbparse which can use just-parsed data to determine dynamically sized data following.

consume_padding(*layer_name, offset*)

Returns the amount of padding used between fields.

Return type

`int`

consume_type(*module, offset, length*)

Returns a (leaf_type, name, object) Tuple for a type, and the number of bytes consumed.

Return type

`Tuple[Tuple[Optional[ObjectInterface], Optional[str], Union[None, List, ObjectInterface]], int]`

property `context`

convert_bytes_to_guid(*original*)

Convert the bytes to the correct ordering for a GUID.

Return type

`str`

convert_fields(*fields*)

Converts a field list into a list of fields.

Return type

`Dict[Optional[str], Dict[str, Any]]`

determine_extended_value(*leaf_type, value, module, length*)

Reads a value and potentially consumes more data to construct the value.

Return type

`Tuple[str, ObjectInterface, int]`

get_json()

Returns the intermediate format JSON data from this pdb file.

get_size_from_index(*index*)

Returns the size of the structure based on the type index provided.

Return type

`int`

get_type_from_index(*index*)

Takes a type index and returns appropriate dictionary.

Return type

`Union[List[Any], Dict[str, Any]]`

classmethod load_pdb_layer(*context, location*)

Loads a PDB file into a layer within the context and returns the name of the new layer.

Note: the context may be changed by this method

Return type

`Tuple[str, ContextInterface]`

name_strip(*name*)

Strips unnecessary components from the start of a symbol name.

omap_lookup(*address*)

Looks up an address using the omap mapping.

static parse_string(*structure, parse_as_pascal=False, size=0*)

Consumes either a c-string or a pascal string depending on the leaf_type.

Return type

`str`

property pdb_layer_name

process_types(*type_references*)

Reads the TPI and symbol streams to populate the reader's variables.

Return type

`None`

read_dbi_stream()

Reads the DBI Stream.

Return type

`None`

read_ipi_stream()

read_necessary_streams()

Read streams to populate the various internal components for a PDB table.

read_pdb_info_stream()

Reads in the pdb information stream.

read_symbol_stream()

Reads in the symbol stream.

read_tpi_stream()

Reads the TPI type steam.

Return type

`None`

replace_forward_references(*types, type_references*)

Finds all ForwardArrayCounts and calculates them once ForwardReferences have been resolved.

reset()

```
type_handlers = {'LF_ARGLIST': ('LF_ENUM', True, None), 'LF_ARRAY': ('LF_ARRAY',
True, 'size'), 'LF_ARRAY_ST': ('LF_ARRAY', True, 'size'), 'LF_BITFIELD':
('LF_BITFIELD', False, None), 'LF_BUILDINFO': ('LF_BUILDINFO', False, None),
'LF_CLASS': ('LF_STRUCTURE', True, 'size'), 'LF_CLASS_ST': ('LF_STRUCTURE', True,
'size'), 'LF_CLASS_VS19': ('LF_STRUCTURE_VS19', True, 'size'), 'LF_ENUM':
('LF_ENUM', True, None), 'LF_ENUMERATE': ('LF_ENUMERATE', True, 'value'),
'LF_FIELDLIST': ('LF_FIELDLIST', False, None), 'LF_FUNC_ID': ('LF_FUNC_ID', True,
None), 'LF_INTERFACE': ('LF_STRUCTURE', True, 'size'), 'LF_MEMBER': ('LF_MEMBER',
True, 'offset'), 'LF_MEMBER_ST': ('LF_MEMBER', True, 'offset'), 'LF_MODIFIER':
('LF_MODIFIER', False, None), 'LF_POINTER': ('LF_POINTER', False, None),
'LF_PROCEDURE': ('LF_PROCEDURE', False, None), 'LF_STRIDED_ARRAY': ('LF_ARRAY',
True, 'size'), 'LF_STRING_ID': ('LF_STRING_ID', True, None), 'LF_STRUCTURE':
('LF_STRUCTURE', True, 'size'), 'LF_STRUCTURE_ST': ('LF_STRUCTURE', True, 'size'),
'LF_STRUCTURE_VS19': ('LF_STRUCTURE_VS19', True, 'size'), 'LF_UDT_MOD_SRC_LINE':
('LF_UDT_MOD_SRC_LINE', False, None), 'LF_UDT_SRC_LINE': ('LF_UDT_SRC_LINE', False,
None), 'LF_UNION': ('LF_UNION', True, None)}
```

class PdbRetreiver

Bases: `object`

retreive_pdb(*guid, file_name, progress_callback=None*)

Return type

`Optional[str]`

volatility3.framework.symbols.windows.pdbutil module

class PDBUtility(*args, **kwargs)

Bases: `VersionableInterface`

Class to handle and manage all getting symbols based on MZ header

classmethod download_pdb_isf(*context, guid, age, pdb_name, progress_callback=None*)

Attempts to download the PDB file, convert it to an ISF file and save it to one of the symbol locations.

Return type

`None`

classmethod `get_guid_from_mz(context, layer_name, offset)`

Takes the offset to an MZ header, locates any available pdb headers, and extracts the guid, age and pdb_name from them

Parameters

- **context** (*ContextInterface*) – The context on which to operate
- **layer_name** (*str*) – The name of the (contiguous) layer within the context that contains the MZ file
- **offset** (*int*) – The offset in the layer at which the MZ file begins

Return type

Optional[*Tuple*[*str*, *int*, *str*]]

Returns

A tuple of the guid, age and pdb_name, or None if no PDB record can be found

classmethod `load_windows_symbol_table(context, guid, age, pdb_name, symbol_table_class, config_path='pdbutility', progress_callback=None)`

Loads (downloading if necessary) a windows symbol table

classmethod `module_from_pdb(context, config_path, layer_name, pdb_name, module_offset=None, module_size=None)`

Creates a module in the specified layer_name based on a pdb name.

Searches the memory section of the loaded module for its PDB GUID and loads the associated symbol table into the symbol space.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **config_path** (*str*) – The config path where to find symbol files
- **layer_name** (*str*) – The name of the layer on which to operate
- **module_offset** (*int*) – This memory dump's module image offset
- **module_size** (*int*) – The size of the module for this dump

Return type

str

Returns

The name of the constructed and loaded symbol table

classmethod `pdbname_scan(ctx, layer_name, page_size, pdb_names, progress_callback=None, start=None, end=None, maximum_invalid_count=100)`

Scans through *layer_name* at *ctx* looking for RSDS headers that indicate one of four common pdb kernel names (as listed in *self.pdb_names*) and returns the tuple (GUID, age, pdb_name, signature_offset, mz_offset) :rtype: *Generator*[*Dict*[*str*, *Union*[*bytes*, *str*, *int*, *None*]], *None*, *None*]

Note: This is automagical and therefore not guaranteed to provide correct results.

The UI should always provide the user an opportunity to specify the appropriate types and PDB values themselves :type layer_name: *str* :param layer_name: The layer name to scan :type page_size: *int* :param page_size: Size of page constant :type pdb_names: *List*[*bytes*] :param pdb_names: List of pdb names to scan :type progress_callback: *Optional*[*Callable*[[*float*, *str*], *None*]] :param progress_callback:

Means of providing the user with feedback during long processes :type start: `Optional[int]` :param start: Start address to start scanning from the pdb_names :type end: `Optional[int]` :param end: Minimum address to scan the pdb_names :type maximum_invalid_count: `int` :param maximum_invalid_count: Amount of pages that can be invalid during scanning before aborting signature search

```
classmethod symbol_table_from_offset(context, layer_name, offset, symbol_table_class='volatility3.framework.symbols.intermed.IntermediateSymbolTable',
                                     config_path=None, progress_callback=None)
```

Produces the name of a symbol table loaded from the offset for an MZ header

Parameters

- **context** (`ContextInterface`) – The context on which to operate
- **layer_name** (`str`) – The name of the (contiguous) layer within the context that contains the MZ file
- **offset** (`int`) – The offset in the layer at which the MZ file begins
- **symbol_table_class** (`str`) – The class to use when constructing the SymbolTable
- **config_path** (`str`) – New path for the produced symbol table configuration with the config tree
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – Callable called to update ongoing progress

Return type

`Optional[str]`

Returns

None if no pdb information can be determined, else returned the name of the loaded symbols for the MZ

```
classmethod symbol_table_from_pdb(context, config_path, layer_name, pdb_name,
                                  module_offset=None, module_size=None)
```

Creates symbol table for a module in the specified layer_name.

Searches the memory section of the loaded module for its PDB GUID and loads the associated symbol table into the symbol space.

Parameters

- **context** (`ContextInterface`) – The context to retrieve required elements (layers, symbol tables) from
- **config_path** (`str`) – The config path where to find symbol files
- **layer_name** (`str`) – The name of the layer on which to operate
- **module_offset** (`int`) – This memory dump's module image offset
- **module_size** (`int`) – The size of the module for this dump

Return type

`str`

Returns

The name of the constructed and loaded symbol table

version = (1, 0, 1)

```
class PdbSignatureScanner(pdb_names)
```

Bases: *ScannerInterface*

A *ScannerInterface* based scanner use to identify Windows PDB records.

Parameters

pdb_names (*List*[*bytes*]) – A list of bytestrings, used to match pdb signatures against the pdb names within the records.

Note: The pdb_names must be a list of byte strings, unicode strs will not match against the data scanned

property context: *ContextInterface* | *None*

property layer_name: *str* | *None*

overlap = 16384

The size of overlap needed for the signature to ensure data cannot hide between two scanned chunks

thread_safe = *True*

Determines whether the scanner accesses global variables in a thread safe manner (for use with *multiprocessing*)

version = (*0*, *0*, *0*)

volatility3.framework.symbols.windows.versions module

```
class OsDistinguisher(version_check, fallback_checks)
```

Bases: *object*

Distinguishes a symbol table as being above a particular version or point.

This will primarily check the version metadata first and foremost. If that metadata isn't available then each item in the fallback_checks is tested. If invert is specified then the result will be true if the version is less than that specified, or in the case of fallback, if any of the fallback checks is successful.

A fallback check is made up of:

- a symbol or type name
- a member name (implying that the value before was a type name)
- whether that symbol, type or member must be present or absent for the symbol table to be more above the required point

Note: Specifying that a member must not be present includes the whole type not being present too (ie, either will pass the test)

Parameters

- **version_check** (*Callable*[[*Tuple*[*int*, ...]], *bool*]) – Function that takes a 4-tuple version and returns whether whether the provided version is above a particular point
- **fallback_checks** (*List*[*Tuple*[*str*, *Optional*[*str*], *bool*]]) – A list of symbol/types/members of types, and whether they must be present to be above the required point

Returns

A function that takes a context and a symbol table name and determines whether that symbol table passes the distinguishing checks

Submodules**volatility3.framework.symbols.intermed module**

class ISFormatTable(*context, config_path, name, json_object, native_types=None, table_mapping=None*)

Bases: *SymbolTableInterface*

Provide a base class to identify all subclasses.

Instantiates an SymbolTable based on an IntermediateSymbolFormat JSON file. This is validated against the appropriate schema.

Parameters

- **context** (*ContextInterface*) – The volatility context for the symbol table
- **config_path** (*str*) – The configuration path for the symbol table
- **name** (*str*) – The name for the symbol table (this is used in symbols e.g. table!symbol)
- **isf_url** – The URL pointing to the ISF file location
- **native_types** (*NativeTableInterface*) – The NativeSymbolTable that contains the native types for this symbol table
- **table_mapping** (*Optional[Dict[str, str]]*) – A dictionary linking names referenced in the file with symbol tables in the context
- **class_types** – A dictionary of type names and classes that override StructType when they are instantiated

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

clear_symbol_cache()

Clears the symbol cache of the symbol table.

Return type

None

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

del_type_class(*name*)

Removes the associated class override for a specific Symbol type.

Return type

`None`

property enumerations: `Iterable[Any]`

Returns an iterator of the Enumeration names.

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

`List[RequirementInterface]`

get_symbol(*name*)

Resolves a symbol name into a symbol object.

If the symbol isn't found, it raises a SymbolError exception

Return type

`SymbolInterface`

get_symbol_type(*name*)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type

`Optional[Template]`

get_symbols_by_location(*offset*, *size=0*)

Returns the name of all symbols in this table that live at a particular offset.

Return type

`Iterable[str]`

get_symbols_by_type(*type_name*)

Returns the name of all symbols in this table that have type matching type_name.

Return type

`Iterable[str]`

get_type(*name*)

Resolves a symbol name into an object template.

If the symbol isn't found it raises a SymbolError exception

Return type

`Template`

get_type_class(*name*)

Returns the class associated with a Symbol type.

Return type

`Type[ObjectInterface]`

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration

- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property metadata: `MetadataInterface` | `None`

Returns a metadata object containing information about the symbol table.

property natives: `NativeTableInterface`

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name*, *clazz*)

Calls the `set_type_class` function but does not throw an exception. Returns whether setting the type class was successful. :type name: `str` :param name: The name of the type to override the class for :type clazz: `Type[ObjectInterface]` :param clazz: The actual class to override for the provided type name

Return type

`bool`

set_type_class(*name*, *clazz*)

Overrides the object class for a specific Symbol type.

Name *must* be present in `self.types`

Parameters

- **name** (`str`) – The name of the type to override the class for
- **clazz** (`Type[ObjectInterface]`) – The actual class to override for the provided type name

Return type

`None`

property symbols: `Iterable[str]`

Returns an iterator of the Symbol names.

property types: `Iterable[str]`

Returns an iterator of the Symbol type names.

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (0, 0, 0)

```
class IntermediateSymbolTable(context, config_path, name, isf_url, native_types=None,
                              table_mapping=None, validate=True, class_types=None, symbol_mask=0)
```

Bases: [SymbolTableInterface](#)

The IntermediateSymbolTable class reads a JSON file and conducts common tasks such as validation, construction by looking up a JSON file from the available files and ensuring the appropriate version of the schema and proxy are chosen.

The JSON format itself is made up of various groups (symbols, user_types, base_types, enums and metadata)

- Symbols link a name to a particular offset relative to the start of a section of memory
- Base types define the simplest primitive data types, these can make more complex structure
- User types define the more complex types by specifying members at a relative offset from the start of the type
- Enums can specify a list of names and values and a type inside which the numeric encoding will fit
- Metadata defines information about the originating file

These are documented in JSONSchema JSON files located in volatility3/schemas.

Instantiates a SymbolTable based on an IntermediateSymbolFormat JSON file. This is validated against the appropriate schema. The validation can be disabled by passing `validate = False`, but this should almost never be done.

Parameters

- **context** ([ContextInterface](#)) – The volatility context for the symbol table
- **config_path** ([str](#)) – The configuration path for the symbol table
- **name** ([str](#)) – The name for the symbol table (this is used in symbols e.g. table!symbol)
- **isf_url** ([str](#)) – The URL pointing to the ISF file location
- **native_types** ([NativeTableInterface](#)) – The NativeSymbolTable that contains the native types for this symbol table
- **table_mapping** ([Optional\[Dict\[str, str\]\]](#)) – A dictionary linking names referenced in the file with symbol tables in the context
- **validate** ([bool](#)) – Determines whether the ISF file will be validated against the appropriate schema
- **class_types** ([Optional\[Mapping\[str, Type\[ObjectInterface\]\]\]](#)) – A dictionary of type names and classes that override StructType when they are instantiated
- **symbol_mask** ([int](#)) – An address mask used for all returned symbol offsets from this table (a mask of 0 disables masking)

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

[HierarchicalDict](#)

clear_symbol_cache(*args, **kwargs)

Clears the symbol cache of this symbol table.

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod create(context, config_path, sub_path, filename, native_types=None, table_mapping=None, class_types=None, symbol_mask=0)

Takes a context and loads an intermediate symbol table based on a filename.

Parameters

- **context** (*ContextInterface*) – The context that the current plugin is being run within
- **config_path** (*str*) – The configuration path for reading/storing configuration information this symbol table may use
- **sub_path** (*str*) – The path under a suitable symbol path (defaults to volatility3/symbols and volatility3/framework/symbols) to check
- **filename** (*str*) – Basename of the file to find under the sub_path
- **native_types** (*Optional[NativeTableInterface]*) – Set of native types, defaults to native types read from the intermediate symbol format file
- **table_mapping** (*Optional[Dict[str, str]]*) – a dictionary of table names mentioned within the ISF file, and the tables within the context which they map to
- **symbol_mask** (*int*) – An address mask used for all returned symbol offsets from this table (a mask of 0 disables masking)

Return type

str

Returns

the name of the added symbol table

del_type_class(*args, **kwargs)

Removes the associated class override for a specific Symbol type.

property enumerations

Returns an iterator of the Enumeration names.

classmethod file_symbol_url(sub_path, filename=None)

Returns an iterator of appropriate file-scheme symbol URLs that can be opened by a ResourceAccessor class.

Filter reduces the number of results returned to only those URLs containing that string

Return type

Generator[str, None, None]

get_enumeration(*args, **kwargs)

classmethod `get_requirements()`

Returns a list of RequirementInterface objects required by this object.

Return type

`List[RequirementInterface]`

get_symbol(*args, **kwargs)

Resolves a symbol name into a symbol object.

If the symbol isn't found, it raises a SymbolError exception

get_symbol_type(name)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type

`Optional[Template]`

get_symbols_by_location(offset, size=0)

Returns the name of all symbols in this table that live at a particular offset.

Return type

`Iterable[str]`

get_symbols_by_type(type_name)

Returns the name of all symbols in this table that have type matching type_name.

Return type

`Iterable[str]`

get_type(*args, **kwargs)

Resolves a symbol name into an object template.

If the symbol isn't found it raises a SymbolError exception

get_type_class(*args, **kwargs)

Returns the class associated with a Symbol type.

classmethod `make_subconfig`(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property metadata

property natives: `NativeTableInterface`

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name*, *clazz*)

Calls the `set_type_class` function but does not throw an exception. Returns whether setting the type class was successful. :type name: `str` :param name: The name of the type to override the class for :type clazz: `Type[ObjectInterface]` :param clazz: The actual class to override for the provided type name

Return type`bool`**set_type_class**(*args, **kwargs)

Overrides the object class for a specific Symbol type.

Name *must* be present in `self.types`

Parameters

- **name** – The name of the type to override the class for
- **clazz** – The actual class to override for the provided type name

property symbols

Returns an iterator of the Symbol names.

property types

Returns an iterator of the Symbol type names.

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`**class Version1Format**(*context*, *config_path*, *name*, *json_object*, *native_types=None*, *table_mapping=None*)

Bases: `ISFormatTable`

Class for storing intermediate debugging data as objects and classes.

Instantiates an SymbolTable based on an IntermediateSymbolFormat JSON file. This is validated against the appropriate schema.

Parameters

- **context** (`ContextInterface`) – The volatility context for the symbol table
- **config_path** (`str`) – The configuration path for the symbol table
- **name** (`str`) – The name for the symbol table (this is used in symbols e.g. table!symbol)
- **isf_url** – The URL pointing to the ISF file location
- **native_types** (`NativeTableInterface`) – The NativeSymbolTable that contains the native types for this symbol table
- **table_mapping** (`Optional[Dict[str, str]]`) – A dictionary linking names referenced in the file with symbol tables in the context

- **class_types** – A dictionary of type names and classes that override StructType when they are instantiated

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

clear_symbol_cache()

Clears the symbol cache of the symbol table.

Return type

None

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

del_type_class(name)

Removes the associated class override for a specific Symbol type.

Return type

None

property enumerations: *Iterable[str]*

Returns an iterator of the available enumerations.

get_enumeration(enum_name)

Resolves an individual enumeration.

Return type

Template

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

List[RequirementInterface]

get_symbol(name)

Returns the location offset given by the symbol name.

Return type

SymbolInterface

get_symbol_type(name)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type

Optional[Template]

get_symbols_by_location(*offset*, *size=0*)

Returns the name of all symbols in this table that live at a particular offset.

Return type

`Iterable[str]`

get_symbols_by_type(*type_name*)

Returns the name of all symbols in this table that have type matching *type_name*.

Return type

`Iterable[str]`

get_type(*type_name*)

Resolves an individual symbol.

Return type

`Template`

get_type_class(*name*)

Returns the class associated with a Symbol type.

Return type

`Type[ObjectInterface]`

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from *kwargs*.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property metadata: `MetadataInterface` | `None`

Returns a metadata object containing information about the symbol table.

property natives: `NativeTableInterface`

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name*, *clazz*)

Calls the `set_type_class` function but does not throw an exception. Returns whether setting the type class was successful. :type name: `str` :param name: The name of the type to override the class for :type clazz: `Type[ObjectInterface]` :param clazz: The actual class to override for the provided type name

Return type

`bool`

set_type_class(*name*, *clazz*)

Overrides the object class for a specific Symbol type.

Name *must* be present in `self.types`

Parameters

- **name** (`str`) – The name of the type to override the class for
- **clazz** (`Type[ObjectInterface]`) – The actual class to override for the provided type name

Return type`None`**property symbols:** `Iterable[str]`

Returns an iterator of the symbol names.

property types: `Iterable[str]`

Returns an iterator of the symbol type names.

classmethod unsatisfied(`context, config_path`)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`**version** = (0, 0, 1)**class Version2Format**(`context, config_path, name, json_object, native_types=None, table_mapping=None`)Bases: `Version1Format`

Class for storing intermediate debugging data as objects and classes.

Instantiates an SymbolTable based on an IntermediateSymbolFormat JSON file. This is validated against the appropriate schema.

Parameters

- **context** (`ContextInterface`) – The volatility context for the symbol table
- **config_path** (`str`) – The configuration path for the symbol table
- **name** (`str`) – The name for the symbol table (this is used in symbols e.g. table!symbol)
- **isf_url** – The URL pointing to the ISF file location
- **native_types** (`NativeTableInterface`) – The NativeSymbolTable that contains the native types for this symbol table
- **table_mapping** (`Optional[Dict[str, str]]`) – A dictionary linking names referenced in the file with symbol tables in the context
- **class_types** – A dictionary of type names and classes that override StructType when they are instantiated

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***clear_symbol_cache()**

Clears the symbol cache of the symbol table.

Return type*None***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

del_type_class(*name*)

Removes the associated class override for a specific Symbol type.

Return type*None***property enumerations:** *Iterable[str]*

Returns an iterator of the available enumerations.

get_enumeration(*enum_name*)

Resolves an individual enumeration.

Return type*Template***classmethod get_requirements()**

Returns a list of RequirementInterface objects required by this object.

Return type*List[RequirementInterface]***get_symbol**(*name*)

Returns the location offset given by the symbol name.

Return type*SymbolInterface***get_symbol_type**(*name*)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type*Optional[Template]***get_symbols_by_location**(*offset*, *size=0*)

Returns the name of all symbols in this table that live at a particular offset.

Return type*Iterable[str]***get_symbols_by_type**(*type_name*)

Returns the name of all symbols in this table that have type matching type_name.

Return type`Iterable[str]`**get_type**(*type_name*)

Resolves an individual symbol.

Return type`Template`**get_type_class**(*name*)

Returns the class associated with a Symbol type.

Return type`Type[ObjectInterface]`**classmethod make_subconfig**(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property metadata:** `MetadataInterface | None`

Returns a metadata object containing information about the symbol table.

property natives: `NativeTableInterface`

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name*, *clazz*)Calls the `set_type_class` function but does not throw an exception. Returns whether setting the type class was successful. :type name: `str` :param name: The name of the type to override the class for :type clazz: `Type[ObjectInterface]` :param clazz: The actual class to override for the provided type name**Return type**`bool`**set_type_class**(*name*, *clazz*)

Overrides the object class for a specific Symbol type.

Name *must* be present in `self.types`**Parameters**

- **name** (`str`) – The name of the type to override the class for
- **clazz** (`Type[ObjectInterface]`) – The actual class to override for the provided type name

Return type`None`

property symbols: `Iterable[str]`

Returns an iterator of the symbol names.

property types: `Iterable[str]`

Returns an iterator of the symbol type names.

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (2, 0, 0)

class Version3Format(*context*, *config_path*, *name*, *json_object*, *native_types*=None, *table_mapping*=None)

Bases: `Version2Format`

Class for storing intermediate debugging data as objects and classes.

Instantiates an SymbolTable based on an IntermediateSymbolFormat JSON file. This is validated against the appropriate schema.

Parameters

- **context** (`ContextInterface`) – The volatility context for the symbol table
- **config_path** (`str`) – The configuration path for the symbol table
- **name** (`str`) – The name for the symbol table (this is used in symbols e.g. table!symbol)
- **isf_url** – The URL pointing to the ISF file location
- **native_types** (`NativeTableInterface`) – The NativeSymbolTable that contains the native types for this symbol table
- **table_mapping** (`Optional[Dict[str, str]]`) – A dictionary linking names referenced in the file with symbol tables in the context
- **class_types** – A dictionary of type names and classes that override StructType when they are instantiated

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

clear_symbol_cache()

Clears the symbol cache of the symbol table.

Return type

`None`

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

del_type_class(*name*)

Removes the associated class override for a specific Symbol type.

Return type

None

property enumerations: *Iterable[str]*

Returns an iterator of the available enumerations.

get_enumeration(*enum_name*)

Resolves an individual enumeration.

Return type

Template

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

List[RequirementInterface]

get_symbol(*name*)

Returns the symbol given by the symbol name.

Return type

SymbolInterface

get_symbol_type(*name*)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type

Optional[Template]

get_symbols_by_location(*offset*, *size=0*)

Returns the name of all symbols in this table that live at a particular offset.

Return type

Iterable[str]

get_symbols_by_type(*type_name*)

Returns the name of all symbols in this table that have type matching type_name.

Return type

Iterable[str]

get_type(*type_name*)

Resolves an individual symbol.

Return type

Template

get_type_class(*name*)

Returns the class associated with a Symbol type.

Return type

`Type[ObjectInterface]`

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property metadata: `MetadataInterface` | `None`

Returns a metadata object containing information about the symbol table.

property natives: `NativeTableInterface`

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name*, *clazz*)

Calls the `set_type_class` function but does not throw an exception. Returns whether setting the type class was successful. :type name: `str` :param name: The name of the type to override the class for :type clazz: `Type[ObjectInterface]` :param clazz: The actual class to override for the provided type name

Return type

`bool`

set_type_class(*name*, *clazz*)

Overrides the object class for a specific Symbol type.

Name *must* be present in `self.types`

Parameters

- **name** (`str`) – The name of the type to override the class for
- **clazz** (`Type[ObjectInterface]`) – The actual class to override for the provided type name

Return type

`None`

property symbols: `Iterable[str]`

Returns an iterator of the symbol names.

property types: `Iterable[str]`

Returns an iterator of the symbol type names.

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (2, 1, 0)

class `Version4Format(context, config_path, name, json_object, native_types=None, table_mapping=None)`

Bases: `Version3Format`

Class for storing intermediate debugging data as objects and classes.

Instantiates an SymbolTable based on an IntermediateSymbolFormat JSON file. This is validated against the appropriate schema.

Parameters

- **context** (`ContextInterface`) – The volatility context for the symbol table
- **config_path** (`str`) – The configuration path for the symbol table
- **name** (`str`) – The name for the symbol table (this is used in symbols e.g. table!symbol)
- **isf_url** – The URL pointing to the ISF file location
- **native_types** (`NativeTableInterface`) – The NativeSymbolTable that contains the native types for this symbol table
- **table_mapping** (`Optional[Dict[str, str]]`) – A dictionary linking names referenced in the file with symbol tables in the context
- **class_types** – A dictionary of type names and classes that override StructType when they are instantiated

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

clear_symbol_cache()

Clears the symbol cache of the symbol table.

Return type

`None`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

del_type_class(*name*)

Removes the associated class override for a specific Symbol type.

Return type

`None`

property enumerations: `Iterable[str]`

Returns an iterator of the available enumerations.

```
format_mapping = {'bool': <class 'volatility3.framework.objects.Boolean'>, 'char':
<class 'volatility3.framework.objects.Char'>, 'float': <class
'volatility3.framework.objects.Float'>, 'int': <class
'volatility3.framework.objects.Integer'>, 'void': <class
'volatility3.framework.objects.Integer'>}
```

get_enumeration(*enum_name*)

Resolves an individual enumeration.

Return type

`Template`

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

`List[RequirementInterface]`

get_symbol(*name*)

Returns the symbol given by the symbol name.

Return type

`SymbolInterface`

get_symbol_type(*name*)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type

`Optional[Template]`

get_symbols_by_location(*offset*, *size=0*)

Returns the name of all symbols in this table that live at a particular offset.

Return type

`Iterable[str]`

get_symbols_by_type(*type_name*)

Returns the name of all symbols in this table that have type matching type_name.

Return type

`Iterable[str]`

get_type(*type_name*)

Resolves an individual symbol.

Return type*Template***get_type_class**(*name*)

Returns the class associated with a Symbol type.

Return type*Type[ObjectInterface]***classmethod make_subconfig**(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property metadata:** *MetadataInterface* | *None*

Returns a metadata object containing information about the symbol table.

property natives: *NativeTableInterface*

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name*, *clazz*)

Calls the set_type_class function but does not throw an exception. Returns whether setting the type class was successful. :type name: *str* :param name: The name of the type to override the class for :type clazz: *Type[ObjectInterface]* :param clazz: The actual class to override for the provided type name

Return type*bool***set_type_class**(*name*, *clazz*)

Overrides the object class for a specific Symbol type.

Name *must* be present in self.types

Parameters

- **name** (*str*) – The name of the type to override the class for
- **clazz** (*Type[ObjectInterface]*) – The actual class to override for the provided type name

Return type*None***property symbols:** *Iterable[str]*

Returns an iterator of the symbol names.

property types: *Iterable[str]*

Returns an iterator of the symbol type names.

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (4, 0, 0)

class `Version5Format(context, config_path, name, json_object, native_types=None, table_mapping=None)`

Bases: `Version4Format`

Class for storing intermediate debugging data as objects and classes.

Instantiates an SymbolTable based on an IntermediateSymbolFormat JSON file. This is validated against the appropriate schema.

Parameters

- **context** (`ContextInterface`) – The volatility context for the symbol table
- **config_path** (`str`) – The configuration path for the symbol table
- **name** (`str`) – The name for the symbol table (this is used in symbols e.g. table!symbol)
- **isf_url** – The URL pointing to the ISF file location
- **native_types** (`NativeTableInterface`) – The NativeSymbolTable that contains the native types for this symbol table
- **table_mapping** (`Optional[Dict[str, str]]`) – A dictionary linking names referenced in the file with symbol tables in the context
- **class_types** – A dictionary of type names and classes that override StructType when they are instantiated

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

clear_symbol_cache()

Clears the symbol cache of the symbol table.

Return type

`None`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

del_type_class(*name*)

Removes the associated class override for a specific Symbol type.

Return type

`None`

property enumerations: `Iterable[str]`

Returns an iterator of the available enumerations.

```
format_mapping = {'bool': <class 'volatility3.framework.objects.Boolean'>, 'char':  
<class 'volatility3.framework.objects.Char'>, 'float': <class  
'volatility3.framework.objects.Float'>, 'int': <class  
'volatility3.framework.objects.Integer'>, 'void': <class  
'volatility3.framework.objects.Integer'>}
```

get_enumeration(*enum_name*)

Resolves an individual enumeration.

Return type

`Template`

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

`List[RequirementInterface]`

get_symbol(*name*)

Returns the symbol given by the symbol name.

Return type

`SymbolInterface`

get_symbol_type(*name*)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type

`Optional[Template]`

get_symbols_by_location(*offset*, *size=0*)

Returns the name of all symbols in this table that live at a particular offset.

Return type

`Iterable[str]`

get_symbols_by_type(*type_name*)

Returns the name of all symbols in this table that have type matching *type_name*.

Return type

`Iterable[str]`

get_type(*type_name*)

Resolves an individual symbol.

Return type*Template***get_type_class**(*name*)

Returns the class associated with a Symbol type.

Return type*Type[ObjectInterface]***classmethod make_subconfig**(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property metadata:** *MetadataInterface* | *None*

Returns a metadata object containing information about the symbol table.

property natives: *NativeTableInterface*

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name*, *clazz*)Calls the set_type_class function but does not throw an exception. Returns whether setting the type class was successful. :type name: *str* :param name: The name of the type to override the class for :type clazz: *Type[ObjectInterface]* :param clazz: The actual class to override for the provided type name**Return type***bool***set_type_class**(*name*, *clazz*)

Overrides the object class for a specific Symbol type.

Name *must* be present in self.types**Parameters**

- **name** (*str*) – The name of the type to override the class for
- **clazz** (*Type[ObjectInterface]*) – The actual class to override for the provided type name

Return type*None***property symbols:** *Iterable[str]*

Returns an iterator of the symbol names.

property types: *Iterable[str]*

Returns an iterator of the symbol type names.

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (4, 1, 0)

class `Version6Format(context, config_path, name, json_object, native_types=None, table_mapping=None)`

Bases: `Version5Format`

Class for storing intermediate debugging data as objects and classes.

Instantiates an SymbolTable based on an IntermediateSymbolFormat JSON file. This is validated against the appropriate schema.

Parameters

- **context** (`ContextInterface`) – The volatility context for the symbol table
- **config_path** (`str`) – The configuration path for the symbol table
- **name** (`str`) – The name for the symbol table (this is used in symbols e.g. table!symbol)
- **isf_url** – The URL pointing to the ISF file location
- **native_types** (`NativeTableInterface`) – The NativeSymbolTable that contains the native types for this symbol table
- **table_mapping** (`Optional[Dict[str, str]]`) – A dictionary linking names referenced in the file with symbol tables in the context
- **class_types** – A dictionary of type names and classes that override StructType when they are instantiated

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

clear_symbol_cache()

Clears the symbol cache of the symbol table.

Return type

`None`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

del_type_class(*name*)

Removes the associated class override for a specific Symbol type.

Return type

`None`

property enumerations: `Iterable[str]`

Returns an iterator of the available enumerations.

```
format_mapping = {'bool': <class 'volatility3.framework.objects.Boolean'>, 'char':
<class 'volatility3.framework.objects.Char'>, 'float': <class
'volatility3.framework.objects.Float'>, 'int': <class
'volatility3.framework.objects.Integer'>, 'void': <class
'volatility3.framework.objects.Integer'>}
```

get_enumeration(*enum_name*)

Resolves an individual enumeration.

Return type

`Template`

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

`List[RequirementInterface]`

get_symbol(*name*)

Returns the symbol given by the symbol name.

Return type

`SymbolInterface`

get_symbol_type(*name*)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type

`Optional[Template]`

get_symbols_by_location(*offset*, *size=0*)

Returns the name of all symbols in this table that live at a particular offset.

Return type

`Iterable[str]`

get_symbols_by_type(*type_name*)

Returns the name of all symbols in this table that have type matching type_name.

Return type

`Iterable[str]`

get_type(*type_name*)

Resolves an individual symbol.

Return type*Template***get_type_class**(*name*)

Returns the class associated with a Symbol type.

Return type*Type[ObjectInterface]***classmethod make_subconfig**(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property metadata:** *MetadataInterface* | *None*

Returns a MetadataInterface object.

property natives: *NativeTableInterface*

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name*, *clazz*)

Calls the set_type_class function but does not throw an exception. Returns whether setting the type class was successful. :type name: *str* :param name: The name of the type to override the class for :type clazz: *Type[ObjectInterface]* :param clazz: The actual class to override for the provided type name

Return type*bool***set_type_class**(*name*, *clazz*)

Overrides the object class for a specific Symbol type.

Name *must* be present in self.types

Parameters

- **name** (*str*) – The name of the type to override the class for
- **clazz** (*Type[ObjectInterface]*) – The actual class to override for the provided type name

Return type*None***property symbols:** *Iterable[str]*

Returns an iterator of the symbol names.

property types: *Iterable[str]*

Returns an iterator of the symbol type names.

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (6, 0, 0)

class `Version7Format(context, config_path, name, json_object, native_types=None, table_mapping=None)`

Bases: `Version6Format`

Class for storing intermediate debugging data as objects and classes.

Instantiates an SymbolTable based on an IntermediateSymbolFormat JSON file. This is validated against the appropriate schema.

Parameters

- **context** (`ContextInterface`) – The volatility context for the symbol table
- **config_path** (`str`) – The configuration path for the symbol table
- **name** (`str`) – The name for the symbol table (this is used in symbols e.g. table!symbol)
- **isf_url** – The URL pointing to the ISF file location
- **native_types** (`NativeTableInterface`) – The NativeSymbolTable that contains the native types for this symbol table
- **table_mapping** (`Optional[Dict[str, str]]`) – A dictionary linking names referenced in the file with symbol tables in the context
- **class_types** – A dictionary of type names and classes that override StructType when they are instantiated

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

clear_symbol_cache()

Clears the symbol cache of the symbol table.

Return type

`None`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

del_type_class(*name*)

Removes the associated class override for a specific Symbol type.

Return type

`None`

property enumerations: `Iterable[str]`

Returns an iterator of the available enumerations.

```
format_mapping = {'bool': <class 'volatility3.framework.objects.Boolean'>, 'char':  
<class 'volatility3.framework.objects.Char'>, 'float': <class  
'volatility3.framework.objects.Float'>, 'int': <class  
'volatility3.framework.objects.Integer'>, 'void': <class  
'volatility3.framework.objects.Integer'>}
```

get_enumeration(*enum_name*)

Resolves an individual enumeration.

Return type

`Template`

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

`List[RequirementInterface]`

get_symbol(*name*)

Returns the symbol given by the symbol name.

Return type

`SymbolInterface`

get_symbol_type(*name*)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type

`Optional[Template]`

get_symbols_by_location(*offset*, *size=0*)

Returns the name of all symbols in this table that live at a particular offset.

Return type

`Iterable[str]`

get_symbols_by_type(*type_name*)

Returns the name of all symbols in this table that have type matching type_name.

Return type

`Iterable[str]`

get_type(*type_name*)

Resolves an individual symbol.

Return type*Template***get_type_class**(*name*)

Returns the class associated with a Symbol type.

Return type*Type[ObjectInterface]***classmethod make_subconfig**(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property metadata:** *MetadataInterface* | *None*

Returns a MetadataInterface object.

property natives: *NativeTableInterface*

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name*, *clazz*)

Calls the set_type_class function but does not throw an exception. Returns whether setting the type class was successful. :type name: *str* :param name: The name of the type to override the class for :type clazz: *Type[ObjectInterface]* :param clazz: The actual class to override for the provided type name

Return type*bool***set_type_class**(*name*, *clazz*)

Overrides the object class for a specific Symbol type.

Name *must* be present in self.types**Parameters**

- **name** (*str*) – The name of the type to override the class for
- **clazz** (*Type[ObjectInterface]*) – The actual class to override for the provided type name

Return type*None***property symbols:** *Iterable[str]*

Returns an iterator of the symbol names.

property types: *Iterable[str]*

Returns an iterator of the symbol type names.

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (6, 1, 0)

class `Version8Format(context, config_path, name, json_object, native_types=None, table_mapping=None)`

Bases: `Version7Format`

Class for storing intermediate debugging data as objects and classes.

Instantiates an SymbolTable based on an IntermediateSymbolFormat JSON file. This is validated against the appropriate schema.

Parameters

- **context** (`ContextInterface`) – The volatility context for the symbol table
- **config_path** (`str`) – The configuration path for the symbol table
- **name** (`str`) – The name for the symbol table (this is used in symbols e.g. table!symbol)
- **isf_url** – The URL pointing to the ISF file location
- **native_types** (`NativeTableInterface`) – The NativeSymbolTable that contains the native types for this symbol table
- **table_mapping** (`Optional[Dict[str, str]]`) – A dictionary linking names referenced in the file with symbol tables in the context
- **class_types** – A dictionary of type names and classes that override StructType when they are instantiated

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

clear_symbol_cache()

Clears the symbol cache of the symbol table.

Return type

`None`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

del_type_class(*name*)

Removes the associated class override for a specific Symbol type.

Return type

`None`

property enumerations: `Iterable[str]`

Returns an iterator of the available enumerations.

```
format_mapping = {'bool': <class 'volatility3.framework.objects.Boolean'>, 'char':
<class 'volatility3.framework.objects.Char'>, 'float': <class
'volatility3.framework.objects.Float'>, 'int': <class
'volatility3.framework.objects.Integer'>, 'void': <class
'volatility3.framework.objects.Integer'>}
```

get_enumeration(*enum_name*)

Resolves an individual enumeration.

Return type

`Template`

classmethod get_requirements()

Returns a list of RequirementInterface objects required by this object.

Return type

`List[RequirementInterface]`

get_symbol(*name*)

Returns the symbol given by the symbol name.

Return type

`SymbolInterface`

get_symbol_type(*name*)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type

`Optional[Template]`

get_symbols_by_location(*offset*, *size=0*)

Returns the name of all symbols in this table that live at a particular offset.

Return type

`Iterable[str]`

get_symbols_by_type(*type_name*)

Returns the name of all symbols in this table that have type matching *type_name*.

Return type

`Iterable[str]`

get_type(*type_name*)

Resolves an individual symbol.

Return type*Template***get_type_class**(*name*)

Returns the class associated with a Symbol type.

Return type*Type[ObjectInterface]***classmethod make_subconfig**(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property metadata:** *MetadataInterface* | *None*

Returns a MetadataInterface object.

property natives: *NativeTableInterface*

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name*, *clazz*)

Calls the set_type_class function but does not throw an exception. Returns whether setting the type class was successful. :type name: *str* :param name: The name of the type to override the class for :type clazz: *Type[ObjectInterface]* :param clazz: The actual class to override for the provided type name

Return type*bool***set_type_class**(*name*, *clazz*)

Overrides the object class for a specific Symbol type.

Name *must* be present in self.types

Parameters

- **name** (*str*) – The name of the type to override the class for
- **clazz** (*Type[ObjectInterface]*) – The actual class to override for the provided type name

Return type*None***property symbols:** *Iterable[str]*

Returns an iterator of the symbol names.

property types: *Iterable[str]*

Returns an iterator of the symbol type names.

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (6, 2, 0)

volatility3.framework.symbols.metadata module

class `LinuxMetadata(json_data)`

Bases: `MetadataInterface`

Class to handle the metadata from a Linux symbol table.

Constructor that accepts json_data.

class `WindowsMetadata(json_data)`

Bases: `MetadataInterface`

Class to handle the metadata from a Windows symbol table.

Constructor that accepts json_data.

property `pdb_age`: `int` | `None`

property `pdb_guid`: `str` | `None`

property `pe_version`: `Tuple[int, int, int]` | `Tuple[int, int, int, int]` | `None`

property `pe_version_string`: `str` | `None`

volatility3.framework.symbols.native module

class `NativeTable(name, native_dictionary)`

Bases: `NativeTableInterface`

Symbol List that handles Native types.

Parameters

- **name** (`str`) – Name of the symbol table
- **native_types** – The native symbol table used to resolve any base/native types
- **table_mapping** – A dictionary mapping names of tables (which when present within the table will be changed to the mapped table)
- **class_types** – A dictionary of types and classes that should be instantiated instead of Struct to construct them

clear_symbol_cache()

Clears the symbol cache of this symbol table.

Return type

None

del_type_class(name)

Removes the associated class override for a specific Symbol type.

Return type

None

property enumerations: *Iterable[str]*

Returns an iterator of the Enumeration names.

get_enumeration(name)**Return type**

Template

get_symbol(name)

Resolves a symbol name into a symbol object.

If the symbol isn't found, it raises a SymbolError exception

Return type

SymbolInterface

get_symbol_type(name)

Resolves a symbol name into a symbol and then resolves the symbol's type.

Return type

Optional[Template]

get_symbols_by_location(offset, size=0)

Returns the name of all symbols in this table that live at a particular offset.

Return type

Iterable[str]

get_symbols_by_type(type_name)

Returns the name of all symbols in this table that have type matching type_name.

Return type

Iterable[str]

get_type(type_name)

Resolves a symbol name into an object template.

This always construct a new python object, rather than using a cached value otherwise changes made later may affect the cached copy. Calling clone after every native type construction was extremely slow.

Return type

Template

get_type_class(name)

Returns the class associated with a Symbol type.

Return type

Type[ObjectInterface]

property natives: [*NativeTableInterface*](#)

Returns None or a NativeTable for handling space specific native types.

optional_set_type_class(*name*, *clazz*)

Calls the set_type_class function but does not throw an exception. Returns whether setting the type class was successful. :type name: [*str*](#) :param name: The name of the type to override the class for :type clazz: [*Type\[ObjectInterface\]*](#) :param clazz: The actual class to override for the provided type name

Return type

[*bool*](#)

set_type_class(*name*, *clazz*)

Overrides the object class for a specific Symbol type.

Name *must* be present in self.types

Parameters

- **name** ([*str*](#)) – The name of the type to override the class for
- **clazz** ([*Type\[ObjectInterface\]*](#)) – The actual class to override for the provided type name

Return type

[*None*](#)

property symbols: [*Iterable\[str\]*](#)

Returns an iterator of the Symbol names.

property types: [*Iterable\[str\]*](#)

Returns an iterator of the symbol type names.

volatility3.framework.symbols.wrappers module

class [**Flags**](#)(*choices*)

Bases: [*object*](#)

Object that converts an integer into a set of flags based on their masks.

property choices: [*ReadOnlyMapping*](#)

Submodules

volatility3.framework.exceptions module

A list of potential exceptions that volatility can throw.

These include exceptions that can be thrown on errors by the symbol space or symbol tables, and by layers when an address is invalid. The [*PagedInvalidAddressException*](#) contains information about the size of the invalid page.

exception [**InvalidAddressException**](#)(*layer_name*, *invalid_address*, *args)

Bases: [*LayerException*](#)

Thrown when an address is not valid in the layer it was requested.

add_note()

Exception.add_note(note) – add a note to the exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception LayerException(layer_name, *args)

Bases: [VolatilityException](#)

Thrown when an error occurs dealing with memory and layers.

add_note()

Exception.add_note(note) – add a note to the exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception MissingModuleException(module, *args)

Bases: [VolatilityException](#)

add_note()

Exception.add_note(note) – add a note to the exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception OfflineException(url, *args)

Bases: [VolatilityException](#)

Throw when a remote resource is requested but Volatility is in offline mode

add_note()

Exception.add_note(note) – add a note to the exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception PagedInvalidAddressException(layer_name, invalid_address, invalid_bits, entry, *args)

Bases: [InvalidAddressException](#)

Thrown when an address is not valid in the paged space in which it was request. This is a subclass of [InvalidAddressException](#) and is only thrown from a paged layer. In most circumstances [InvalidAddressException](#) is the correct exception to throw, since this will catch all invalid mappings (including paged ones).

Includes the invalid address and the number of bits of the address that are invalid

add_note()

Exception.add_note(note) – add a note to the exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception PluginRequirementExceptionBases: *VolatilityException*

Class to allow plugins to indicate that a requirement has not been fulfilled.

add_note()

Exception.add_note(note) – add a note to the exception

args**with_traceback()**

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception PluginVersionExceptionBases: *VolatilityException*

Class to allow determining that a required plugin has an invalid version.

add_note()

Exception.add_note(note) – add a note to the exception

args**with_traceback()**

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception SwappedInvalidAddressException(*layer_name, invalid_address, invalid_bits, entry, swap_offset, *args*)Bases: *PagedInvalidAddressException*

Thrown when an address is not valid in the paged layer in which it was requested, but expected to be in an associated swap layer.

Includes the swap lookup, as well as the invalid address and the bits of the lookup that were invalid.

add_note()

Exception.add_note(note) – add a note to the exception

args**with_traceback()**

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception SymbolError(*symbol_name, table_name, *args*)Bases: *VolatilityException*

Thrown when a symbol lookup has failed.

add_note()

Exception.add_note(note) – add a note to the exception

args**with_traceback()**

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception SymbolSpaceErrorBases: *VolatilityException*

Thrown when an error occurs dealing with Symbolspaces and SymbolTables.

add_note()

Exception.add_note(note) – add a note to the exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception UnsatisfiedException(unsatisfied)

Bases: [VolatilityException](#)

add_note()

Exception.add_note(note) – add a note to the exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception VolatilityException

Bases: [Exception](#)

Class to allow filtering of all VolatilityExceptions.

add_note()

Exception.add_note(note) – add a note to the exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

10.1.3 volatility3.plugins package

Defines the plugin architecture.

This is the namespace for all volatility plugins, and determines the path for loading plugins

NOTE: This file is important for core plugins to run (which certain components such as the windows registry layers) are dependent upon, please DO NOT alter or remove this file unless you know the consequences of doing so.

The framework is configured this way to allow plugin developers/users to override any plugin functionality whether existing or new.

Subpackages

volatility3.plugins.linux package

All Linux-related plugins.

NOTE: This file is important for core plugins to run (which certain components such as the windows registry layers) are dependent upon, please DO NOT alter or remove this file unless you know the consequences of doing so.

The framework is configured this way to allow plugin developers/users to override any plugin functionality whether existing or new.

When overriding the plugins directory, you must include a file like this in any subdirectories that may be necessary.

Submodules

volatility3.plugins.linux.bash module

A module containing a collection of plugins that produce data typically found in Linux's /proc file system.

class `Bash(context, config_path, progress_callback=None)`

Bases: `PluginInterface`, `TimeLinerInterface`

Recovers bash command history from memory.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

`build_configuration()`

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

`generate_timeline()`

Method generates Tuples of (description, timestamp_type, timestamp)

These need not be generated in any particular order, sorting will be done later

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.plugins.linux.capabilities module

class Capabilities(context, config_path, progress_callback=None)

Bases: *PluginInterface*

Lists process capabilities

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional*[Callable[[float, str], None]]) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod get_task_capabilities(task)

Returns a tuple with the task basic information along with its capabilities

Parameters

task (*ObjectInterface*) – A task object from where to get the fields.

Return type

Tuple[TaskData, CapabilitiesData]

Returns

A tuple with the task basic information and its capabilities

classmethod get_tasks_capabilities(tasks)

Yields a tuple for each task containing the task's basic information along with its capabilities

Parameters

tasks (*List[ObjectInterface]*) – An iterable with the tasks to process.

Yields

A tuple for each task containing the task's basic information and its capabilities

Return type

Iterable[Tuple[TaskData, CapabilitiesData]]

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (1, 0, 0)

class CapabilitiesData(cap_inheritable, cap_permitted, cap_effective, cap_bset, cap_ambient)

Bases: `object`

Stores each set of capabilities for a task

astuple()

Returns a shallow copy of the capability sets in a tuple.

Otherwise, when `dataclasses.astuple()` performs a deep-copy recursion on `ObjectInterface` will take a substantial amount of time.

Return type

`Tuple`

cap_ambient: `ObjectInterface`

cap_bset: `ObjectInterface`

```
cap_effective: ObjectInterface
cap_inheritable: ObjectInterface
cap_permitted: ObjectInterface
```

```
class TaskData(comm, pid, tgid, ppid, euid)
```

```
Bases: object
```

Stores basic information about a task

```
comm: str
```

```
euid: int
```

```
pid: int
```

```
ppid: int
```

```
tgid: int
```

volatility3.plugins.linux.check_afinfo module

A module containing a collection of plugins that produce data typically found in Linux's /proc file system.

```
class Check_afinfo(context, config_path, progress_callback=None)
```

```
Bases: PluginInterface
```

Verifies the operation function pointers of network protocols.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

```
build_configuration()
```

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

```
HierarchicalDict
```

```
property config: HierarchicalDict
```

The Hierarchical configuration Dictionary for this Configurable object.

```
property config_path: str
```

The configuration path on which this configurable lives.

```
property context: ContextInterface
```

The context object that this configurable belongs to/configuration is stored in.

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property `open`

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects `self.validate` to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.plugins.linux.check_creds module**class** `Check_creds(context, config_path, progress_callback=None)`Bases: `PluginInterface`

Checks if any processes are sharing credential structures

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type`HierarchicalDict`**property config:** `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.linux.check_idt module**class Check_idt(context, config_path, progress_callback=None)**

Bases: *PluginInterface*

Checks if the IDT has been altered

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

`version = (0, 0, 0)`

volatility3.plugins.linux.check_modules module

class `Check_modules(context, config_path, progress_callback=None)`

Bases: `PluginInterface`

Compares module list to sysfs info, if available

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod `get_kset_modules(context, vmlinux_name)`

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.linux.check_syscall module

A module containing a collection of plugins that produce data typically found in Linux's /proc file system.

class Check_syscall(context, config_path, progress_callback=None)

Bases: *PluginInterface*

Check system call table for hooks.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.plugins.linux.elfs module

A module containing a collection of plugins that produce data typically found in Linux's /proc file system.

class Elfs(context, config_path, progress_callback=None)

Bases: `PluginInterface`

Lists all memory mapped ELF files for all processes.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too.

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: [ContextInterface](#)

The context object that this configurable belongs to/configuration is stored in.

classmethod elf_dump(*context, layer_name, elf_table_name, vma, task, open_method*)

Extracts an ELF as a FileHandlerInterface :type context: [ContextInterface](#) :param context: the context to operate upon :type layer_name: [str](#) :param layer_name: The name of the layer on which to operate :type elf_table_name: [str](#) :param elf_table_name: the name for the symbol table containing the symbols for ELF-files :type vma: [ObjectInterface](#) :param vma: virtual memory allocation of ELF :type task: [ObjectInterface](#) :param task: the task object whose memory should be output :type open_method: [Type\[FileHandlerInterface\]](#) :param open_method: class to provide context manager for opening the file

Return type

[Optional\[FileHandlerInterface\]](#)

Returns

An open FileHandlerInterface object containing the complete data for the task or None in the case of failure

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

[List\[RequirementInterface\]](#)

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** ([ContextInterface](#)) – The context in which to store the new configuration
- **base_config_path** ([str](#)) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

[str](#)

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (2, 0, 0)

volatility3.plugins.linux.envvars module

class Envvars(*context, config_path, progress_callback=None*)

Bases: *PluginInterface*

Lists processes with their environment variables

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.linux.iomem module

class `IOMem(context, config_path, progress_callback=None)`

Bases: `PluginInterface`

Generates an output similar to /proc/iomem on a running system.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

classmethod `parse_resource(context, vmlinux_module_name, resource_offset, seen={}, depth=0)`

Recursively parse from a root resource to find details about all related resources.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **vmlinux_module_name** (*str*) – The name of the kernel module on which to operate
- **resource_offset** (*int*) – The offset to the resource to be parsed
- **seen** (*set*) – The set of resource offsets that have already been parsed
- **depth** (*int*) – How deep into the resource structure we are

Yields

Each row of output

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 1)

volatility3.plugins.linux.keyboard_notifiers module**class Keyboard_notifiers**(*context, config_path, progress_callback=None*)Bases: *PluginInterface*

Parses the keyboard notifier call chain

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.linux.kmsg module**class ABCKmsg(context, config)**

Bases: ABC

Kernel log buffer reader

FACILITIES = ('kern', 'user', 'mail', 'daemon', 'auth', 'syslog', 'lpr', 'news', 'uucp', 'cron', 'authpriv', 'ftp')

LEVELS = ('emerg', 'alert', 'crit', 'err', 'warn', 'notice', 'info', 'debug')

get_caller(obj)

get_caller_text(caller_id)

classmethod get_facility_text(facility)

Return type

str

classmethod get_level_text(level)

Return type

str

get_prefix(*obj*)

Return type

`Tuple[int, int, str, str]`

get_string(*addr, length*)

Return type

`str`

get_timestamp_in_sec_str(*obj*)

Return type

`str`

nsec_to_sec_str(*nsec*)

Return type

`str`

abstract run()

Walks through the specific kernel implementation.

Return type

`Iterator[Tuple[str, str, str, str, str]]`

classmethod run_all(*context, config*)

It calls each subclass `syntab_checks()` to test the required conditions to that specific kernel implementation.

Parameters

- **context** (*ContextInterface*) – The volatility3 context on which to operate
- **config** (*HierarchicalDict*) – Core configuration

Yields

kmsg records

Return type

`Iterator[Tuple[str, str, str, str, str]]`

abstract classmethod syntab_checks(*vmlinux*)

This method on each subclass will be called to evaluate if the kernel being analyzed fulfill the type & symbols requirements for the implementation. The first class returning True will be instantiated and called via the `run()` method.

Return type

`bool`

Returns

True is the kernel being analysed fulfill the class requirements.

class DescStateEnum(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `Enum`

desc_committed = 1

desc_finalized = 2

desc_miss = -1

```
desc_reserved = 0
```

```
desc_reusable = 3
```

```
class Kmsg(context, config_path, progress_callback=None)
```

Bases: [*PluginInterface*](#)

Kernel log buffer reader

Parameters

- **context** ([*ContextInterface*](#)) – The context that the plugin will operate within
- **config_path** ([*str*](#)) – The path to configuration data within the context configuration data
- **progress_callback** ([*Optional*](#)[[*Callable*](#)[[[*float*](#), [*str*](#)], [*None*](#)]]) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

[*HierarchicalDict*](#)

property config: [*HierarchicalDict*](#)

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: [*str*](#)

The configuration path on which this configurable lives.

property context: [*ContextInterface*](#)

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

[*List*](#)[[*RequirementInterface*](#)]

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** ([*ContextInterface*](#)) – The context in which to store the new configuration
- **base_config_path** ([*str*](#)) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

[*str*](#)

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 1)

class KmsgFiveTen(context, config)

Bases: *ABCKmsg*

In 5.10 the kernel ringbuffer implementation changed. Previously only one process should read /proc/kmsg and it is permanently open and periodically read by the syslog daemon. A high level structure 'printk_ringbuffer' was added to represent the printk ringbuffer which actually contains two ringbuffers. The descriptor ring 'desc_ring' contains the records' metadata, text offsets and states. The data block ring 'text_data_ring' contains the records' text strings. A pointer to the high level structure is kept in the prb pointer which is initialized to a static ringbuffer.

```
static struct printk_ringbuffer *prb = &printk_rb_static;
```

In SMP systems with more than 64 CPUs this ringbuffer size is dynamically allocated according the number of CPUs based on the value of CONFIG_LOG_CPU_MAX_BUF_SHIFT. The prb pointer is updated consequently to this dynamic ringbuffer in setup_log_buf().

```
prb = &printk_rb_dynamic;
```

Behind scenes, log_buf is still used as external buffer. When the static printk_ringbuffer struct is initialized, _DEFINE_PRINTKRB sets text_data_ring.data pointer to the address in log_buf which points to the static buffer __log_buff. If a dynamic ringbuffer takes place, setup_log_buf() sets text_data_ring.data of printk_rb_dynamic to the new allocated external buffer via the prb_init function. In that case, the original external static buffer in __log_buf and printk_rb_static are unused.

```
new_log_buf = memblock_alloc(new_log_buf_len, LOG_ALIGN);
prb_init(&printk_rb_dynamic, new_log_buf, ...);
log_buf = new_log_buf;
prb = &printk_rb_dynamic;
```

See `printk.c` and `printk_ringbuffer.c` in `kernel/printk/` folder for more details.

```
FACILITIES = ('kern', 'user', 'mail', 'daemon', 'auth', 'syslog', 'lpr', 'news',
'uucp', 'cron', 'authpriv', 'ftp')
```

```
LEVELS = ('emerg', 'alert', 'crit', 'err', 'warn', 'notice', 'info', 'debug')
```

`get_caller(obj)`

`get_caller_text(caller_id)`

`get_dict_lines(info)`

Return type

`Generator[str, None, None]`

`classmethod get_facility_text(facility)`

Return type

`str`

`classmethod get_level_text(level)`

Return type

`str`

`get_log_lines(text_data_ring, desc, info)`

Return type

`Generator[str, None, None]`

`get_prefix(obj)`

Return type

`Tuple[int, int, str, str]`

`get_string(addr, length)`

Return type

`str`

`get_text_from_data_ring(text_data_ring, desc, info)`

Return type

`str`

`get_timestamp_in_sec_str(obj)`

Return type

`str`

`nsec_to_sec_str(nsec)`

Return type

`str`

run()

Walks through the specific kernel implementation.

Return type

`Iterator[Tuple[str, str, str, str, str]]`

classmethod run_all(context, config)

It calls each subclass `syntab_checks()` to test the required conditions to that specific kernel implementation.

Parameters

- **context** (`ContextInterface`) – The volatility3 context on which to operate
- **config** (`HierarchicalDict`) – Core configuration

Yields

kmsg records

Return type

`Iterator[Tuple[str, str, str, str, str]]`

classmethod syntab_checks(vmlinux)

This method on each subclass will be called to evaluate if the kernel being analyzed fulfill the type & symbols requirements for the implementation. The first class returning True will be instantiated and called via the `run()` method.

Return type

`bool`

Returns

True is the kernel being analysed fulfill the class requirements.

class KmsgLegacy(context, config)

Bases: `ABCKmsg`

Linux kernels prior to v5.10, the ringbuffer is initially kept in `__log_buf`, and `log_buf` is a pointer to the former. `__log_buf` is declared as a char array but it actually contains an array of `printk_log` structs. The length of this array is defined in the kernel `KConfig` configuration via the `CONFIG_LOG_BUF_SHIFT` value as a power of 2. This can also be modified by the `log_buf_len` kernel boot parameter. In SMP systems with more than 64 CPUs this ringbuffer size is dynamically allocated according the number of CPUs based on the value of `CONFIG_LOG_CPU_MAX_BUF_SHIFT`, and the `log_buf` pointer is updated consequently to the new buffer. In that case, the original static buffer in `__log_buf` is unused.

```
FACILITIES = ('kern', 'user', 'mail', 'daemon', 'auth', 'syslog', 'lpr', 'news',
              'uucp', 'cron', 'authpriv', 'ftp')
```

```
LEVELS = ('emerg', 'alert', 'crit', 'err', 'warn', 'notice', 'info', 'debug')
```

```
get_caller(obj)
```

```
get_caller_text(caller_id)
```

```
get_dict_lines(msg)
```

Return type

`Generator[str, None, None]`

classmethod get_facility_text(facility)**Return type**

`str`

classmethod `get_level_text(level)`

Return type

`str`

get_log_lines(msg)

Return type

`Generator[str, None, None]`

get_prefix(obj)

Return type

`Tuple[int, int, str, str]`

get_string(addr, length)

Return type

`str`

get_text_from_printk_log(msg)

Return type

`str`

get_timestamp_in_sec_str(obj)

Return type

`str`

nsec_to_sec_str(nsec)

Return type

`str`

run()

Walks through the specific kernel implementation.

Return type

`Iterator[Tuple[str, str, str, str, str]]`

classmethod `run_all(context, config)`

It calls each subclass `syntab_checks()` to test the required conditions to that specific kernel implementation.

Parameters

- **context** (*ContextInterface*) – The volatility3 context on which to operate
- **config** (*HierarchicalDict*) – Core configuration

Yields

kmsg records

Return type

`Iterator[Tuple[str, str, str, str, str]]`

classmethod `syntab_checks(vmlinux)`

This method on each subclass will be called to evaluate if the kernel being analyzed fulfill the type & symbols requirements for the implementation. The first class returning True will be instantiated and called via the `run()` method.

Return type

`bool`

Returns

True is the kernel being analysed fulfill the class requirements.

volatility3.plugins.linux.Lsmod module

A module containing a collection of plugins that produce data typically found in Linux's /proc file system.

class Lsmod(*context, config_path, progress_callback=None*)

Bases: *PluginInterface*

Lists loaded kernel modules.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod list_modules(*context, vmlinux_module_name*)

Lists all the modules in the primary layer.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** – The name of the layer on which to operate
- **vmlinux_symbols** – The name of the table containing the kernel symbols

Yields

The modules present in the *layer_name* layer's modules list

Return type`Iterable[ObjectInterface]`

This function will throw a `SymbolError` exception if kernel module support is not enabled.

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from `kwargs`.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property open**

Returns a context manager and thus can be called like `open`

run()

Executes the functionality of the code.

Note: This method expects `self.validate` to have been called to ensure all necessary options have been provided

Returns

A `TreeGrid` object that can then be passed to a `Renderer`.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type`None`

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return `[]`, it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`

version = (2, 0, 0)

volatility3.plugins.linux.lsof module

A module containing a collection of plugins that produce data typically found in Linux's /proc file system.

class Lsof(*context, config_path, progress_callback=None*)

Bases: *PluginInterface*

Lists all memory maps for all processes.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod list_fds(*context, symbol_table, filter_func=<function Lsof.<lambda>>*)

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 1, 0)

volatility3.plugins.linux.malfind module**class Malfind(context, config_path, progress_callback=None)**

Bases: *PluginInterface*

Lists process memory ranges that potentially contain injected code.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type*List[RequirementInterface]***classmethod make_subconfig**(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type*None***classmethod unsatisfied**(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

`version = (0, 0, 0)`

volatility3.plugins.linux.mountinfo module

class MountInfo(context, config_path, progress_callback=None)

Bases: `PluginInterface`

Lists mount points on processes mount namespaces

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_mountinfo(mnt, task)

Extract various information about a mount point. It mimics the Linux kernel show_mountinfo function.

Return type

`Optional[Tuple[int, int, str, str, str, List[str], List[str], str, str, List[str]]]`

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property `open`

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 0)

class `MountInfoData(mnt_id, parent_id, st_dev, mnt_root_path, path_root, mnt_opts, fields, mnt_type, devname, sb_opts)`

Bases: *tuple*

Create new instance of MountInfoData(mnt_id, parent_id, st_dev, mnt_root_path, path_root, mnt_opts, fields, mnt_type, devname, sb_opts)

count(*value*, /)

Return number of occurrences of value.

devname

Alias for field number 8

fields

Alias for field number 6

index(*value*, *start*=0, *stop*=9223372036854775807, /)

Return first index of value.

Raises ValueError if the value is not present.

mnt_id

Alias for field number 0

mnt_opts

Alias for field number 5

mnt_root_path

Alias for field number 3

mnt_type

Alias for field number 7

parent_id

Alias for field number 1

path_root

Alias for field number 4

sb_opts

Alias for field number 9

st_dev

Alias for field number 2

volatility3.plugins.linux.proc module

A module containing a collection of plugins that produce data typically found in Linux's /proc file system.

class **Maps**(*context*, *config_path*, *progress_callback*=None)

Bases: [*PluginInterface*](#)

Lists all memory maps for all processes.

Parameters

- **context** ([*ContextInterface*](#)) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional*[*Callable*[[*float*, *str*], None]]) – A callable that can provide feedback at progress points

MAXSIZE_DEFAULT = 1073741824

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod list_vmas(*task*, *filter_func*=<function Maps.<lambda>>)

Lists the Virtual Memory Areas of a specific process.

Parameters

- **task** (*ObjectInterface*) – task object from which to list the vma
- **filter_func** (*Callable*[[*ObjectInterface*], *bool*]) – Function to take a vma and return False if it should be filtered out

Return type

Generator[[*ObjectInterface*, *None*, *None*]

Returns

Yields vmas based on the task and filtered based on the filter function

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (1, 0, 0)**classmethod vma_dump(context, task, vm_start, vm_end, open_method, maxsize=1073741824)**

Extracts the complete data for VMA as a FileInterface.

Parameters

- **context** (`ContextInterface`) – The context to retrieve required elements (layers, symbol tables) from
- **task** (`ObjectInterface`) – an task_struct instance
- **vm_start** (`int`) – The start virtual address from the vma to dump
- **vm_end** (`int`) – The end virtual address from the vma to dump
- **open_method** (`Type[FileHandlerInterface]`) – class to provide context manager for opening the file
- **maxsize** (`int`) – Max size of VMA section (default MAXSIZE_DEFAULT)

Return type

`Optional[FileHandlerInterface]`

Returns

An open FileInterface object containing the complete data for the task or None in the case of failure

volatility3.plugins.linux.psaux module**class PsAux**(*context, config_path, progress_callback=None*)Bases: *PluginInterface*

Lists processes with their command line arguments

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.linux.pslist module**class PsList(context, config_path, progress_callback=None)**

Bases: *PluginInterface*

Lists the processes present in a particular linux memory image.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod create_pid_filter(*pid_list=None*)

Constructs a filter function for process IDs.

Parameters

pid_list (*List[int]*) – List of process IDs that are acceptable (or None if all are acceptable)

Return type

Callable[[Any], bool]

Returns

Function which, when provided a process object, returns True if the process is to be filtered out of the list

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod get_task_fields(*task, decorate_comm=False*)

Extract the fields needed for the final output :type task: *ObjectInterface* :param task: A task object from where to get the fields. :type decorate_comm: *bool* :param decorate_comm:

If True, it decorates the comm string of

- User threads: in curly brackets,
- Kernel threads: in square brackets

Defaults to False.

Return type

Tuple[int, int, int, str]

Returns

A tuple with the fields to show in the plugin output.

classmethod list_tasks(*context, vmlinux_module_name, filter_func=<function PsList.<lambda>>, include_threads=False*)

Lists all the tasks in the primary layer.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **vmlinux_module_name** (*str*) – The name of the kernel module on which to operate
- **filter_func** (*Callable[[int], bool]*) – A function which takes a process object and returns True if the process should be ignored/filtered
- **include_threads** (*bool*) – If True, it will also return user threads.

Yields

Task objects

Return type

`Iterable[ObjectInterface]`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects `self.validate` to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (2, 2, 0)

volatility3.plugins.linux.psscan module

class DescExitStateEnum(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: [Enum](#)

Enum for linux task exit_state as defined in include/linux/sched.h

EXIT_DEAD = 16

EXIT_TRACE = 48

EXIT_ZOMBIE = 32

TASK_RUNNING = 0

class PsScan(*context, config_path, progress_callback=None*)

Bases: [PluginInterface](#)

Scans for processes present in a particular linux image.

Parameters

- **context** ([ContextInterface](#)) – The context that the plugin will operate within
- **config_path** ([str](#)) – The path to configuration data within the context configuration data
- **progress_callback** ([Optional](#)[[Callable](#)[[[float](#), [str](#)], [None](#)]]) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

[HierarchicalDict](#)

property config: [HierarchicalDict](#)

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: [str](#)

The configuration path on which this configurable lives.

property context: [ContextInterface](#)

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

[List](#)[[RequirementInterface](#)]

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** ([ContextInterface](#)) – The context in which to store the new configuration

- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

classmethod scan_tasks(*context, vmlinux_module_name, kernel_layer_name*)

Scans for tasks in the memory layer.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **vmlinux_module_name** (*str*) – The name of the kernel module on which to operate
- **kernel_layer_name** (*str*) – The name for the kernel layer

Yields

Task objects

Return type

Iterable[ObjectInterface]

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

```
version = (1, 0, 0)
```

volatility3.plugins.linux.pstree module

class `PsTree(context, config_path, progress_callback=None)`

Bases: `PluginInterface`

Plugin for listing processes in a tree based on their parent process ID.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

`build_configuration()`

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

`find_level(pid)`

Finds how deep the PID is in the tasks hierarchy.

Parameters

- **pid** (`int`) – PID to find the level in the hierarchy

Return type

`None`

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.plugins.linux.sockstat module**class SockHandlers(vmlinux, task)**

Bases: `VersionableInterface`

Handles several socket families extracting the sockets information.

process_sock(sock)

Takes a kernel generic *sock* object and processes it with its respective socket family

Parameters

sock (`StructType`) – Kernel generic *sock* object

Return type

`Tuple[StructType, Tuple[str, str, str], Dict]`

Returns a tuple with:

sock: The respective kernel's `*_sock` object for that socket family
 sock_stat: A tuple with the source and destination (address and port) along with its state string
 socket_filter: A dictionary with information about the socket filter

version = (1, 0, 0)

class Sockstat(context, config_path, progress_callback=None)

Bases: [PluginInterface](#)

Lists all network connections for all processes.

Parameters

- **context** ([ContextInterface](#)) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

[HierarchicalDict](#)

property config: [HierarchicalDict](#)

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: [ContextInterface](#)

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod list_sockets(context, symbol_table, filter_func=<function Sockstat.<lambda>>)

Returns every single socket descriptor

Parameters

- **context** ([ContextInterface](#)) – The context to retrieve required elements (layers, symbol tables) from
- **symbol_table** (`str`) – The name of the kernel module on which to operate
- **filter_func** (`Callable[[int], bool]`) – A function which takes a task object and returns True if the task should be ignored/filtered

Yields

task – Kernel's task object
 netns_id: Network namespace ID
 fd_num: File descriptor number
 family: Socket family string (AF_UNIX, AF_INET, etc)
 sock_type: Socket type string (STREAM, DGRAM, etc)
 protocol: Protocol string (UDP, TCP, etc)
 sock_fields: A tuple with the `*_sock` object, the sock stats and the extended info dictionary

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 0)

volatility3.plugins.linux.tty_check module**class** `tty_check(context, config_path, progress_callback=None)`Bases: `PluginInterface`

Checks tty devices for hooks

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type`HierarchicalDict`**property config:** `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type`List[RequirementInterface]`**classmethod make_subconfig(context, base_config_path, **kwargs)**

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.linux.vmayarascan module**class VmaYaraScan(context, config_path, progress_callback=None)**

Bases: *PluginInterface*

Scans all virtual memory areas for tasks using yara.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

static get_vma_maps(task)

Creates a map of start/end addresses for each virtual memory area in a task.

Parameters

task (*ObjectInterface*) – The task object of which to read the vmas from

Return type

Iterable[Tuple[int, int]]

Returns

An iterable of tuples containing start and end addresses for each descriptor

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod **unsatisfied**(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (1, 0, 0)

volatility3.plugins.mac package

All Mac-related plugins.

NOTE: This file is important for core plugins to run (which certain components such as the windows registry layers) are dependent upon, please DO NOT alter or remove this file unless you know the consequences of doing so.

The framework is configured this way to allow plugin developers/users to override any plugin functionality whether existing or new.

When overriding the plugins directory, you must include a file like this in any subdirectories that may be necessary.

Submodules

volatility3.plugins.mac.bash module

A module containing a collection of plugins that produce data typically found in mac's /proc file system.

class **Bash**(*context, config_path, progress_callback=None*)

Bases: `PluginInterface`, `TimeLinerInterface`

Recovers bash command history from memory.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

generate_timeline()

Method generates Tuples of (description, timestamp_type, timestamp)

These need not be generated in any particular order, sorting will be done later

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type*None*

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.plugins.mac.check_syscall module

class `Check_syscall(context, config_path, progress_callback=None)`

Bases: `PluginInterface`

Check system call table for hooks.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.mac.check_sysctl module**class** `Check_sysctl(context, config_path, progress_callback=None)`Bases: `PluginInterface`

Check sysctl handlers for hooks.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type`HierarchicalDict`**property config:** `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type`List[RequirementInterface]`**classmethod make_subconfig(context, base_config_path, **kwargs)**

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.mac.check_trap_table module**class Check_trap_table(context, config_path, progress_callback=None)**

Bases: *PluginInterface*

Check mach trap table for hooks.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```

unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))

```

Return type

`Dict[str, RequirementInterface]`

`version = (0, 0, 0)`

volatility3.plugins.mac.ifconfig module

class Ifconfig(context, config_path, progress_callback=None)

Bases: [*PluginInterface*](#)

Lists network interface information for all devices

Parameters

- **context** ([*ContextInterface*](#)) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

[*HierarchicalDict*](#)

property config: [*HierarchicalDict*](#)

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: [*ContextInterface*](#)

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** ([*ContextInterface*](#)) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.plugins.mac.kauth_listeners module

class Kauth_listeners(context, config_path, progress_callback=None)

Bases: *PluginInterface*

Lists kauth listeners and their status

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional*[Callable[[float, str], None]]) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.plugins.mac.kauth_scopes module

class `Kauth_scopes(context, config_path, progress_callback=None)`

Bases: `PluginInterface`

Lists kauth scopes and their status

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

classmethod `list_kauth_scopes(context, kernel_module_name, filter_func=<function Kauth_scopes.<lambda>>)`

Enumerates the registered kauth scopes and yields each object Uses smear-safe enumeration API

Return type

`Iterable[ObjectInterface]`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (2, 0, 0)

volatility3.plugins.mac.kevents module**class Kevents**(*context, config_path, progress_callback=None*)Bases: *PluginInterface*

Lists event handlers registered by processes

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

```
all_filters = {4: [('NOTE_DELETE', 1), ('NOTE_WRITE', 2), ('NOTE_EXTEND', 4),
('NOTE_ATTRIB', 8), ('NOTE_LINK', 16), ('NOTE_RENAME', 32), ('NOTE_REVOKE', 64)], 5:
[('NOTE_EXIT', 2147483648), ('NOTE_EXITSTATUS', 67108864), ('NOTE_FORK',
1073741824), ('NOTE_EXEC', 536870912), ('NOTE_SIGNAL', 134217728), ('NOTE_REAP',
268435456)], 7: [('NOTE_SECONDS', 1), ('NOTE_USECONDS', 2), ('NOTE_NSECONDS', 4),
('NOTE_ABSOLUTE', 8)]}
```

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

```
event_types = {1: 'EVFILT_READ', 2: 'EVFILT_WRITE', 3: 'EVFILT_AIO', 4:
'EVFILT_VNODE', 5: 'EVFILT_PROC', 6: 'EVFILT_SIGNAL', 7: 'EVFILT_TIMER', 8:
'EVFILT_MACHPORT', 9: 'EVFILT_FS', 10: 'EVFILT_USER', 12: 'EVFILT_VM'}
```

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod list_kernel_events(*context, kernel_module_name, filter_func=<function Kevents.<lambda>>*)

Returns the kernel event filters registered

Return type*Iterable[Tuple[ObjectInterface, ObjectInterface, ObjectInterface]]***Return values:**

A tuple of 3 elements:

- 1) The name of the process that registered the filter
- 2) The process ID of the process that registered the filter
- 3) The object of the associated kernel event filter

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

```
proc_filters = [('NOTE_EXIT', 2147483648), ('NOTE_EXITSTATUS', 67108864),
('NOTE_FORK', 1073741824), ('NOTE_EXEC', 536870912), ('NOTE_SIGNAL', 134217728),
('NOTE_REAP', 268435456)]
```

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

```
timer_filters = [('NOTE_SECONDS', 1), ('NOTE_USECONDS', 2), ('NOTE_NSECONDS', 4),
('NOTE_ABSOLUTE', 8)]
```

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]``version = (1, 0, 0)``vnode_filters = [('NOTE_DELETE', 1), ('NOTE_WRITE', 2), ('NOTE_EXTEND', 4),
('NOTE_ATTRIB', 8), ('NOTE_LINK', 16), ('NOTE_RENAME', 32), ('NOTE_REVOKE', 64)]`**volatility3.plugins.mac.list_files module****class** `List_Files(context, config_path, progress_callback=None)`Bases: `PluginInterface`

Lists all open file descriptors for all processes.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type`HierarchicalDict`**property config:** `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod list_files(context, kernel_module_name)**Return type**`Iterable[ObjectInterface]`**classmethod make_subconfig(context, base_config_path, **kwargs)**

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration

- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.mac.lsmmod module

A module containing a collection of plugins that produce data typically found in Mac's lsmmod command.

class Lsmmod(*context, config_path, progress_callback=None*)

Bases: *PluginInterface*

Lists loaded kernel modules.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within

- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod list_modules(*context, darwin_module_name*)

Lists all the modules in the primary layer.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** – The name of the layer on which to operate
- **darwin_symbols** – The name of the table containing the kernel symbols

Returns

A list of modules from the *layer_name* layer

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (2, 0, 0)

volatility3.plugins.mac.lsof module**class Lsof(context, config_path, progress_callback=None)**

Bases: *PluginInterface*

Lists all open file descriptors for all processes.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: [*HierarchicalDict*](#)

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: [*str*](#)

The configuration path on which this configurable lives.

property context: [*ContextInterface*](#)

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** ([*ContextInterface*](#)) – The context in which to store the new configuration
- **base_config_path** ([*str*](#)) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

[*str*](#)

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

[*None*](#)

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]``version = (0, 0, 0)`**volatility3.plugins.mac.malfind module****class** `Malfind(context, config_path, progress_callback=None)`Bases: `PluginInterface`

Lists process memory ranges that potentially contain injected code.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too.

Return type`HierarchicalDict`**property config:** `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(`context, base_config_path, **kwargs`)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.mac.mount module

A module containing a collection of plugins that produce data typically found in Mac's mount command.

class Mount(context, config_path, progress_callback=None)

Bases: [PluginInterface](#)

A module containing a collection of plugins that produce data typically found in Mac's mount command

Parameters

- **context** ([ContextInterface](#)) – The context that the plugin will operate within
- **config_path** ([str](#)) – The path to configuration data within the context configuration data
- **progress_callback** ([Optional](#)[[Callable](#)[[[float](#), [str](#)], [None](#)]]) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod list_mounts(*context, kernel_module_name*)

Lists all the mount structures in the primary layer.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** – The name of the layer on which to operate
- **darwin_symbols** – The name of the table containing the kernel symbols

Returns

A list of mount structures from the *layer_name* layer

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod **unsatisfied**(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (2, 0, 0)

volatility3.plugins.mac.netstat module

class **Netstat**(*context*, *config_path*, *progress_callback*=None)

Bases: `PluginInterface`

Lists all network connections for all processes.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property **config**: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property **config_path**: `str`

The configuration path on which this configurable lives.

property **context**: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod **get_requirements**()

Returns a list of Requirement objects for this plugin.

classmethod `list_sockets(context, kernel_module_name, filter_func=<function Netstat.<lambda>>)`

Returns the open socket descriptors of a process

Return type

`Iterable[Tuple[ObjectInterface, ObjectInterface, ObjectInterface]]`

Return values:

A tuple of 3 elements:

- 1) The name of the process that opened the socket
- 2) The process ID of the process that opened the socket
- 3) The address of the associated socket structure

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property `open`

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects `self.validate` to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

`version = (0, 0, 0)`

volatility3.plugins.mac.proc_maps module

class `Maps(context, config_path, progress_callback=None)`

Bases: `PluginInterface`

Lists process memory ranges that potentially contain injected code.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too.

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.plugins.mac.psaux module

In-memory artifacts from OSX systems.

class Psaux(context, config_path, progress_callback=None)

Bases: *PluginInterface*

Recovers program command line arguments.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Return type

TreeGrid

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.mac.pslist module

class PsList(*context, config_path, progress_callback=None*)

Bases: *PluginInterface*

Lists the processes present in a particular mac memory image.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod `create_pid_filter(pid_list=None)`

Return type

`Callable[[int], bool]`

classmethod `get_list_tasks(method)`

Returns the `list_tasks` method based on the selector

Parameters

method (`str`) – Must be one fo the available methods in `get_task_choices`

Return type

`Callable[[ContextInterface, str, Callable[[int], bool]], Iterable[ObjectInterface]]`

Returns

`list_tasks` method for listing tasks

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

classmethod `list_tasks_allproc(context, kernel_module_name, filter_func=<function PsList.<lambda>>)`

Lists all the processes in the primary layer based on the `allproc` method

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **kernel_module_name** (`str`) – The name of the the kernel module on which to operate
- **filter_func** (`Callable[[int], bool]`) – A function which takes a process object and returns True if the process should be ignored/filtered

Return type

`Iterable[ObjectInterface]`

Returns

The list of process objects from the processes linked list after filtering

classmethod `list_tasks_pid_hash_table(context, kernel_module_name, filter_func=<function PsList.<lambda>>)`

Lists all the tasks in the primary layer using the pid hash table

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **kernel_module_name** (`str`) – The name of the the kernel module on which to operate
- **filter_func** (`Callable[[int], bool]`) – A function which takes a task object and returns True if the task should be ignored/filtered

Return type

`Iterable[ObjectInterface]`

Returns

The list of task objects from the *layer_name* layer's *tasks* list after filtering


```
classmethod list_tasks_process_group(context, kernel_module_name, filter_func=<function
                                   PsList.<lambda>>>)
```

Lists all the tasks in the primary layer using process groups

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **kernel_module_name** (*str*) – The name of the the kernel module on which to operate
- **filter_func** (*Callable[[int], bool]*) – A function which takes a task object and returns True if the task should be ignored/filtered

Return type

Iterable[ObjectInterface]

Returns

The list of task objects from the *layer_name* layer's *tasks* list after filtering

```
classmethod list_tasks_sessions(context, kernel_module_name, filter_func=<function
                                PsList.<lambda>>>)
```

Lists all the tasks in the primary layer using sessions

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **kernel_module_name** (*str*) – The name of the the kernel module on which to operate
- **filter_func** (*Callable[[int], bool]*) – A function which takes a task object and returns True if the task should be ignored/filtered

Return type

Iterable[ObjectInterface]

Returns

The list of task objects from the *layer_name* layer's *tasks* list after filtering

```
classmethod list_tasks_tasks(context, kernel_module_name, filter_func=<function
                              PsList.<lambda>>>)
```

Lists all the tasks in the primary layer based on the tasks queue

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **kernel_module_name** (*str*) – The name of the the kernel module on which to operate
- **filter_func** (*Callable[[int], bool]*) – A function which takes a task object and returns True if the task should be ignored/filtered

Return type

Iterable[ObjectInterface]

Returns

The list of task objects from the *layer_name* layer's *tasks* list after filtering

```
classmethod make_subconfig(context, base_config_path, **kwargs)
```

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

pslist_methods = ['tasks', 'allproc', 'process_group', 'sessions', 'pid_hash_table']

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (3, 0, 0)

volatility3.plugins.mac.pstree module**class PsTree(*args, **kwargs)**Bases: *PluginInterface*

Plugin for listing processes in a tree based on their parent process ID.

Parameters

- **context** – The context that the plugin will operate within
- **config_path** – The path to configuration data within the context configuration data
- **progress_callback** – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.mac.socket_filters module**class Socket_filters(context, config_path, progress_callback=None)**

Bases: *PluginInterface*

Enumerates kernel socket filters.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

`version = (0, 0, 0)`

volatility3.plugins.mac.timers module

class `Timers(context, config_path, progress_callback=None)`

Bases: `PluginInterface`

Check for malicious kernel timers.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration

- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.mac.trustedbsd module

class Trustedbsd(context, config_path, progress_callback=None)

Bases: *PluginInterface*

Checks for malicious trustedbsd modules

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data

- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects `self.validate` to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.mac.vfsevents module

class VFSevents(*context, config_path, progress_callback=None*)

Bases: *PluginInterface*

Lists processes that are filtering file system events

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

```
event_types = ['CREATE_FILE', 'DELETE', 'STAT_CHANGED', 'RENAME',
'CONTENT_MODIFIED', 'EXCHANGE', 'FINDER_INFO_CHANGED', 'CREATE_DIR', 'CHOWN',
'XATTR_MODIFIED', 'XATTR_REMOVED', 'DOCID_CREATED', 'DOCID_CHANGED']
```

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property `open`

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.windows package

All Windows OS plugins.

NOTE: This file is important for core plugins to run (which certain components such as the windows registry layers) are dependent upon, please DO NOT alter or remove this file unless you know the consequences of doing so.

The framework is configured this way to allow plugin developers/users to override any plugin functionality whether existing or new.

When overriding the plugins directory, you must include a file like this in any subdirectories that may be necessary.

Subpackages

volatility3.plugins.windows.registry package

Windows registry plugins.

NOTE: This file is important for core plugins to run (which certain components such as the windows registry layers) are dependent upon, please DO NOT alter or remove this file unless you know the consequences of doing so.

The framework is configured this way to allow plugin developers/users to override any plugin functionality whether existing or new.

When overriding the plugins directory, you must include a file like this in any subdirectories that may be necessary.

Submodules

volatility3.plugins.windows.registry.hivelist module

class `HiveGenerator`(*cmhive*, *forward=True*)

Bases: `object`

Walks the registry HiveList linked list in a given direction and stores an invalid offset if it's unable to fully walk the list

property `invalid`: `int` | `None`

class `HiveList`(*context*, *config_path*, *progress_callback=None*)

Bases: `PluginInterface`

Lists the registry hives present in a particular memory image.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type*List[RequirementInterface]***classmethod list_hive_objects**(*context, layer_name, symbol_table, filter_string=None*)

Lists all the hives in the primary layer.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols
- **filter_string** (*str*) – A string which must be present in the hive name if specified

Return type*Iterator[ObjectInterface]***Returns**

The list of registry hives from the *layer_name* layer as filtered against using the *filter_string*

classmethod list_hives(*context, base_config_path, layer_name, symbol_table, filter_string=None, hive_offsets=None*)

Walks through a registry, hive by hive returning the constructed registry layer name.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **base_config_path** (*str*) – The configuration path for any settings required by the new table
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols
- **filter_string** (*Optional[str]*) – An optional string which must be present in the hive name if specified
- **offset** – An optional offset to specify a specific hive to iterate over (takes precedence over *filter_string*)

Yields

A registry hive layer name

Return type`Iterable[RegistryHive]`**classmethod** `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property** `open`

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects `self.validate` to have been called to ensure all necessary options have been provided

Return type`TreeGrid`**Returns**

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type`None`**classmethod** `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`**version** = (1, 0, 0)

volatility3.plugins.windows.registry.hivescan module**class HiveScan**(context, config_path, progress_callback=None)Bases: *PluginInterface*

Scans for registry hives present in a particular windows memory image.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too.

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

classmethod scan_hives(context, layer_name, symbol_table)

Scans for hives using the poolscanner module and constraints or bigpools module with tag.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols

Return type

Iterable[ObjectInterface]

Returns

A list of Hive objects as found from the *layer_name* layer based on Hive pool signatures

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 0)

volatility3.plugins.windows.registry.printkey module**class** **PrintKey**(*context, config_path, progress_callback=None*)Bases: *PluginInterface*

Lists the registry keys under a hive or specific key value.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too.

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type*List[RequirementInterface]***classmethod key_iterator**(*hive, node_path=None, recurse=False*)

Walks through a set of nodes from a given node (last one in node_path). Avoids loops by not traversing into nodes already present in the node_path.

Parameters

- **hive** (*RegistryHive*) – The registry hive to walk
- **node_path** (*Sequence[StructType]*) – The list of nodes that make up the
- **recurse** (*bool*) – Traverse down the node tree or stay only on the same level

Yields

A tuple of results (depth, is_key, last write time, path, volatile, and the node).

Return type*Iterable[Tuple[int, bool, datetime, str, bool, ObjectInterface]]***classmethod make_subconfig**(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 0)

volatility3.plugins.windows.registry.userassist module**class UserAssist**(*args, **kwargs)Bases: *PluginInterface*

Print userassist registry keys and information.

Parameters

- **context** – The context that the plugin will operate within
- **config_path** – The path to configuration data within the context configuration data
- **progress_callback** – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type*List[RequirementInterface]***list_userassist**(hive)

Generate userassist data for a registry hive.

Return type*Generator[Tuple[int, Tuple], None, None]***classmethod make_subconfig**(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property open**

Returns a context manager and thus can be called like open

parse_userassist_data(*reg_val*)

Reads the raw data of a `_CM_KEY_VALUE` and returns a dict of userassist fields.

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type`None`**classmethod unsatisfied**(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`

version = (0, 0, 0)

Submodules

volatility3.plugins.windows.bigpools module

class `BigPools`(*context*, *config_path*, *progress_callback*=None)

Bases: `PluginInterface`

List big page pools.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod list_big_pools(*context, layer_name, symbol_table, tags=None, show_free=False*)

Returns the big page pool objects from the kernel PoolBigPageTable array.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols
- **tags** (*Optional[list]*) – An optional list of pool tags to filter big page pool tags by

Yields

A big page pool object

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 1, 0)

volatility3.plugins.windows.cachedump module**class Cachedump(context, config_path, progress_callback=None)**

Bases: *PluginInterface*

Dumps lsa secrets from memory

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

static decrypt_hash(*edata, nlkm, ch, xp*)

static get_nlkm(*sechive, lsakey, is_vista_or_later*)

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

static parse_cache_entry(*cache_data*)

Return type

Tuple[int, int, int, bytes, bytes]

static parse_decrypted_cache(*dec_data, uname_len, domain_len, domain_name_len*)

Get the data from the cache and separate it into the username, domain name, and hash data

Return type

Tuple[str, str, str, bytes]

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 0)

volatility3.plugins.windows.callbacks module

class Callbacks(*context, config_path, progress_callback=None*)

Bases: *PluginInterface*

Lists kernel callbacks and notification routines.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

static `create_callback_table(context, symbol_table, config_path)`

Creates a symbol table for a set of callbacks.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **symbol_table** (*str*) – The name of an existing symbol table containing the kernel symbols
- **config_path** (*str*) – The configuration path within the context of the symbol table to create

Return type

str

Returns

The name of the constructed callback table

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod `list_bugcheck_callbacks(context, layer_name, symbol_table, callback_table_name)`

Lists all kernel bugcheck callbacks.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols
- **callback_table_name** (*str*) – The name of the table containing the callback symbols

Yields

A name, location and optional detail string

Return type

Iterable[Tuple[str, int, str]]

classmethod `list_bugcheck_reason_callbacks(context, layer_name, symbol_table, callback_table_name)`

Lists all kernel bugcheck reason callbacks.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols
- **callback_table_name** (*str*) – The name of the table containing the callback symbols

Yields

A name, location and optional detail string

Return type`Iterable[Tuple[str, int, str]]`**classmethod** `list_notify_routines(context, layer_name, symbol_table, callback_table_name)`

Lists all kernel notification routines.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols
- **callback_table_name** (*str*) – The name of the table containing the callback symbols

Yields

A name, location and optional detail string

Return type`Iterable[Tuple[str, int, Optional[str]]]`**classmethod** `list_registry_callbacks(context, layer_name, symbol_table, callback_table_name)`

Lists all registry callbacks.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols
- **callback_table_name** (*str*) – The name of the table containing the callback symbols

Yields

A name, location and optional detail string

Return type`Iterable[Tuple[str, int, Optional[str]]]`**classmethod** `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property** `open`

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 0)

volatility3.plugins.windows.cmdline module**class CmdLine(context, config_path, progress_callback=None)**

Bases: *PluginInterface*

Lists process command line arguments.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: [*HierarchicalDict*](#)

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: [*str*](#)

The configuration path on which this configurable lives.

property context: [*ContextInterface*](#)

The context object that this configurable belongs to/configuration is stored in.

classmethod get_cmdline(*context, kernel_table_name, proc*)

Extracts the cmdline from PEB

Parameters

- **context** ([*ContextInterface*](#)) – the context to operate upon
- **kernel_table_name** ([*str*](#)) – the name for the symbol table containing the kernel’s symbols
- **proc** – the process object

Returns

A string with the command line

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

[*List\[RequirementInterface\]*](#)

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** ([*ContextInterface*](#)) – The context in which to store the new configuration
- **base_config_path** ([*str*](#)) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

[*str*](#)

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod **unsatisfied**(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (1, 0, 0)

volatility3.plugins.windows.crashinfo module

class **Crashinfo**(*context*, *config_path*, *progress_callback*=None)

Bases: `PluginInterface`

Lists the information from a Windows crash dump.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property **config**: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property **config_path**: `str`

The configuration path on which this configurable lives.

property **context**: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod **get_requirements**()

Returns a list of Requirement objects for this plugin.

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.windows.devicetree module**class DeviceTree**(*context, config_path, progress_callback=None*)Bases: *PluginInterface*

Listing tree based on drivers and attached devices in a particular windows memory image.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type*List[RequirementInterface]***classmethod make_subconfig**(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Return type

TreeGrid

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 1)

volatility3.plugins.windows.dlllist module**class DllList(context, config_path, progress_callback=None)**

Bases: *PluginInterface*, *TimeLinerInterface*

Lists the loaded modules in a particular windows memory image.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod dump_pe(*context, pe_table_name, dll_entry, open_method, layer_name=None, prefix=""*)

Extracts the complete data for a process as a FileInterface

Parameters

- **context** (*ContextInterface*) – the context to operate upon
- **pe_table_name** (*str*) – the name for the symbol table containing the PE format symbols
- **dll_entry** (*ObjectInterface*) – the object representing the module
- **layer_name** (*str*) – the layer that the DLL lives within
- **open_method** (*Type[FileHandlerInterface]*) – class for constructing output files

Return type*Optional[FileHandlerInterface]***Returns**

An open FileHandlerInterface object containing the complete data for the DLL or None in the case of failure

generate_timeline()

Method generates Tuples of (description, timestamp_type, timestamp)

These need not be generated in any particular order, sorting will be done later

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type*List[RequirementInterface]***classmethod make_subconfig**(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str*

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (2, 0, 0)

volatility3.plugins.windows.driverirp module

class DriverIrp(context, config_path, progress_callback=None)

Bases: *PluginInterface*

List IRPs for drivers in a particular windows memory image.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type*None***classmethod unsatisfied**(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```

unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))

```

Return type`Dict[str, RequirementInterface]``version = (0, 0, 0)`**volatility3.plugins.windows.drivermodule module****class DriverModule**(context, config_path, progress_callback=None)Bases: `PluginInterface`

Determines if any loaded drivers were hidden by a rootkit

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type`HierarchicalDict`**property config:** `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type`List[RequirementInterface]`**classmethod make_subconfig**(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration

- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Return type

TreeGrid

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 0)

volatility3.plugins.windows.driverscan module**class DriverScan**(context, config_path, progress_callback=None)Bases: *PluginInterface*

Scans for drivers present in a particular windows memory image.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional*[Callable[[float, str], None]]) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_names_for_driver(driver)

Convenience method for getting the commonly used names associated with a driver

Parameters**driver** – A Eriver object**Returns**

A tuple of strings of (driver name, service key, driver alt. name)

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

classmethod scan_drivers(*context, layer_name, symbol_table*)

Scans for drivers using the poolscanner module and constraints.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (`str`) – The name of the layer on which to operate
- **symbol_table** (`str`) – The name of the table containing the kernel symbols

Return type`Iterable[ObjectInterface]`**Returns**

A list of Driver objects as found from the *layer_name* layer based on Driver pool signatures

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type`None`**classmethod unsatisfied**(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`

version = (1, 0, 0)

volatility3.plugins.windows.dumpfiles module

class `DumpFiles(context, config_path, progress_callback=None)`

Bases: `PluginInterface`

Dumps cached file contents from Windows memory samples.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod dump_file_producer(`file_object, memory_object, open_method, layer, desired_file_name`)

Produce a file from the memory object's `get_available_pages()` interface.

Parameters

- **file_object** (`ObjectInterface`) – the parent `_FILE_OBJECT`
- **memory_object** (`ObjectInterface`) – the `_CONTROL_AREA` or `_SHARED_CACHE_MAP`
- **open_method** (`Type[FileHandlerInterface]`) – class for constructing output files
- **layer** (`DataLayerInterface`) – the memory layer to read from
- **desired_file_name** (`str`) – name of the output file

Return type

`Optional[FileHandlerInterface]`

Returns

result status

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

classmethod `process_file_object(context, primary_layer_name, open_method, file_obj)`

Given a FILE_OBJECT, dump data to separate files for each of the three file caches.

Parameters

- **context** (*ContextInterface*) – the context to operate upon
- **primary_layer_name** (*str*) – primary/virtual layer to operate on
- **open_method** (*Type[FileHandlerInterface]*) – class for constructing output files
- **file_obj** (*ObjectInterface*) – the FILE_OBJECT

Return type

Generator[Tuple, None, None]

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:


```

unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))

```

Return type*Dict[str, RequirementInterface]***version** = (1, 0, 0)**volatility3.plugins.windows.envvars module****class** **Envvars**(*context, config_path, progress_callback=None*)Bases: *PluginInterface*

Display process environment variables

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod **get_requirements()**

Returns a list of Requirement objects for this plugin.

Return type*List[RequirementInterface]***classmethod** **make_subconfig**(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration

- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 0)

volatility3.plugins.windows.filescan module

class FileScan(context, config_path, progress_callback=None)

Bases: *PluginInterface*

Scans for file objects present in a particular windows memory image.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data

- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

classmethod `scan_files(context, layer_name, symbol_table)`

Scans for file objects using the poolscanner module and constraints.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols

Return type

Iterable[ObjectInterface]

Returns

A list of File objects as found from the *layer_name* layer based on File pool signatures

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.windows.getservicesids module

class `GetServiceSIDs(*args, **kwargs)`

Bases: *PluginInterface*

Lists process token sids.

Parameters

- **context** – The context that the plugin will operate within
- **config_path** – The path to configuration data within the context configuration data
- **progress_callback** – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type*List[RequirementInterface]***classmethod make_subconfig**(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type*None***classmethod unsatisfied**(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (1, 0, 0)

createservicesid(svc)

Calculate the Service SID

Return type

`str`

volatility3.plugins.windows.getsids module

class GetSIDs(*args, **kwargs)

Bases: `PluginInterface`

Print the SIDs owning each process

Parameters

- **context** – The context that the plugin will operate within
- **config_path** – The path to configuration data within the context configuration data
- **progress_callback** – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

lookup_user_sids()

Enumerate the registry for all the users.

Returns

user name}

Return type

An dictionary of {sid

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

```
version = (1, 0, 0)
```

```
find_sid_re(sid_string, sid_re_list)
```

Return type

`Union[str, BaseAbsentValue]`

volatility3.plugins.windows.handles module

```
class Handles(*args, **kwargs)
```

Bases: `PluginInterface`

Lists process open handles.

Parameters

- **context** – The context that the plugin will operate within
- **config_path** – The path to configuration data within the context configuration data
- **progress_callback** – A callable that can provide feedback at progress points

```
build_configuration()
```

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

```
classmethod find_cookie(context, layer_name, symbol_table)
```

Find the ObHeaderCookie value (if it exists)

Return type

`Optional[ObjectInterface]`

```
find_sar_value()
```

Locate ObpCaptureHandleInformationEx if it exists in the sample.

Once found, parse it for the SAR value that we need to decode pointers in the `_HANDLE_TABLE_ENTRY` which allows us to find the associated `_OBJECT_HEADER`.

```
classmethod get_requirements()
```

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod `get_type_map(context, layer_name, symbol_table)`

List the executive object types (`_OBJECT_TYPE`) using the `ObTypeIndexTable` or `ObpObjectTypes` symbol (differs per OS). This method will be necessary for determining what type of object we have given an object header.

Note: The object type index map was hard coded into profiles in previous versions of volatility. It is now generated dynamically.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols

Return type

Dict[int, str]

Returns

A mapping of type indices to type names

handles(*handle_table*)

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 0)

volatility3.plugins.windows.hashdump module

class Hashdump(*context, config_path, progress_callback=None*)

Bases: *PluginInterface*

Dumps user hashes from memory

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

almpassword = b'LMPASSWORD\x00'

antpassword = b'NTPASSWORD\x00'

anum = b'0123456789012345678901234567890123456789\x00'

aqwerty = b'!@#\$\$%^&*()qwertyUIOPAzxcvbnmQQQQQQQQQQ)(*@&\x00'

bootkey_perm_table = [8, 5, 4, 2, 11, 9, 13, 3, 0, 6, 1, 12, 14, 10, 15, 7]

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod decrypt_single_hash(*rid, hbootkey, enc_hash, lmntstr*)

classmethod decrypt_single_saltred_hash(*rid, hbootkey, enc_hash, _lmntstr, salt*)

Return type

`Optional[bytes]`

`empty_lm = b'\xaa\xd3\xb45\xb5\x14\x04\xee\xaa\xd3\xb45\xb5\x14\x04\xee'`

`empty_nt = b'!\xd6\xcf\xe0\xd1j\xe91\xb7<Y\xd7\xe0\xc0\x89\xc0'`

classmethod get_bootkey(*syshive*)

Return type

`Optional[bytes]`

classmethod get_hbootkey(*samhive, bootkey*)

Return type

`Optional[bytes]`

classmethod get_hive_key(*hive, key*)

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod get_user_hashes(*user, samhive, hbootkey*)

Return type

`Optional[Tuple[bytes, bytes]]`

classmethod get_user_keys(*samhive*)

Return type

`List[ObjectInterface]`

classmethod get_user_name(*user, samhive*)

Return type

`Optional[bytes]`

`lmkey = b'KGS!@#$$'`

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

```
odd_parity = [1, 1, 2, 2, 4, 4, 7, 7, 8, 8, 11, 11, 13, 13, 14, 14, 16, 16, 19, 19,
21, 21, 22, 22, 25, 25, 26, 26, 28, 28, 31, 31, 32, 32, 35, 35, 37, 37, 38, 38, 41,
41, 42, 42, 44, 44, 47, 47, 49, 49, 50, 50, 52, 52, 55, 55, 56, 56, 59, 59, 61, 61,
62, 62, 64, 64, 67, 67, 69, 69, 70, 70, 73, 73, 74, 74, 76, 76, 79, 79, 81, 81, 82,
82, 84, 84, 87, 87, 88, 88, 91, 91, 93, 93, 94, 94, 97, 97, 98, 98, 100, 100, 103,
103, 104, 104, 107, 107, 109, 109, 110, 110, 112, 112, 115, 115, 117, 117, 118, 118,
121, 121, 122, 122, 124, 124, 127, 127, 128, 128, 131, 131, 133, 133, 134, 134, 137,
137, 138, 138, 140, 140, 143, 143, 145, 145, 146, 146, 148, 148, 151, 151, 152, 152,
155, 155, 157, 157, 158, 158, 161, 161, 162, 162, 164, 164, 167, 167, 168, 168, 171,
171, 173, 173, 174, 174, 176, 176, 179, 179, 181, 181, 182, 182, 185, 185, 186, 186,
188, 188, 191, 191, 193, 193, 194, 194, 196, 196, 199, 199, 200, 200, 203, 203, 205,
205, 206, 206, 208, 208, 211, 211, 213, 213, 214, 214, 217, 217, 218, 218, 220, 220,
223, 223, 224, 224, 227, 227, 229, 229, 230, 230, 233, 233, 234, 234, 236, 236, 239,
239, 241, 241, 242, 242, 244, 244, 247, 247, 248, 248, 251, 251, 253, 253, 254, 254]
```

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod sid_to_key(sid)

Takes rid of a user and converts it to a key to be used by the DES cipher

Return type

`Tuple[bytes, bytes]`

classmethod sidbytes_to_key(s)

Builds final DES key from the strings generated in *sid_to_key*

Return type

`bytes`

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```

unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))

```

Return type

`Dict[str, RequirementInterface]`

`version = (1, 1, 0)`

volatility3.plugins.windows.info module

class Info(*context, config_path, progress_callback=None*)

Bases: *PluginInterface*

Show OS & kernel details of the memory sample being analyzed.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_depends(*context, layer_name, index=0*)

List the dependencies of a given layer.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required layers from
- **layer_name** (*str*) – the name of the starting layer
- **index** (*int*) – the index/order of the layer

Return type

Iterable[Tuple[int, DataLayerInterface]]

Returns

An iterable containing the levels and layer objects for all dependent layers

classmethod `get_kdbg_structure(context, config_path, layer_name, symbol_table)`

Returns the KDDEBUGGER_DATA64 structure for a kernel

Return type

ObjectInterface

classmethod `get_kernel_module(context, layer_name, symbol_table)`

Returns the kernel module based on the layer and symbol_table

classmethod `get_kuser_structure(context, layer_name, symbol_table)`

Returns the _KUSER_SHARED_DATA structure for a kernel

Return type

ObjectInterface

classmethod `get_ntheadr_structure(context, config_path, layer_name)`

Gets the ntheadr structure for the kernel of the specified layer

Return type

ObjectInterface

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod `get_version_structure(context, layer_name, symbol_table)`

Returns the KdVersionBlock information from a kernel

Return type

ObjectInterface

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 0)

volatility3.plugins.windows.joblinks module**class JobLinks(context, config_path, progress_callback=None)**

Bases: *PluginInterface*

Print process job link information

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: HierarchicalDict

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Return type

`TreeGrid`

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:


```

unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))

```

Return type

`Dict[str, RequirementInterface]`

`version = (1, 0, 0)`

volatility3.plugins.windows.ldrmodules module

class LdrModules(*context, config_path, progress_callback=None*)

Bases: *PluginInterface*

Lists the loaded modules in a particular windows memory image.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too.

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (1, 0, 0)

volatility3.plugins.windows.lsadump module

class Lsadump(context, config_path, progress_callback=None)

Bases: *PluginInterface*

Dumps lsa secrets from memory

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional*[Callable[[float, str], None]]) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod decrypt_aes(*secret, key*)

Based on code from <http://lab.mediaservice.net/code/cachedump.rb>

Return type

bytes

classmethod decrypt_secret(*secret, key*)

Python implementation of SystemFunction005.

Decrypts a block of data with DES using given key. Note that key can be longer than 7 bytes.

classmethod get_lsa_key(*sechive, bootkey, vista_or_later*)**Return type**

Optional[bytes]

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod get_secret_by_name(*sechive, name, lsa_key, is_vista_or_later*)**classmethod make_subconfig**(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 0)

volatility3.plugins.windows.malfind module**class Malfind(context, config_path, progress_callback=None)**

Bases: *PluginInterface*

Lists process memory ranges that potentially contain injected code.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod is_vad_empty(*proc_layer, vad*)

Check if a VAD region is either entirely unavailable due to paging, entirely consisting of zeros, or a combination of the two. This helps ignore false positives whose VAD flags match task._injection_filter requirements but there's no data and thus not worth reporting it.

Parameters

- **proc_layer** – the process layer
- **vad** – the MMVAD structure to test

Returns

A boolean indicating whether a vad is empty or not

classmethod list_injections(*context, kernel_layer_name, symbol_table, proc*)

Generate memory regions for a process that may contain injected code.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **kernel_layer_name** (*str*) – The name of the kernel layer from which to read the VAD protections
- **symbol_table** (*str*) – The name of the table containing the kernel symbols
- **proc** (*ObjectInterface*) – an _EPROCESS instance

Return type

Iterable[Tuple[ObjectInterface, bytes]]

Returns

An iterable of VAD instances and the first 64 bytes of data containing in that region

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type`None`**classmethod unsatisfied(context, config_path)**

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.plugins.windows.mbrscan module

class MBRScan(context, config_path, progress_callback=None)

Bases: `PluginInterface`

Scans for and parses potential Master Boot Records (MBRs)

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_hash(data)**Return type**

str

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Return type

TreeGrid

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (1, 0, 0)

volatility3.plugins.windows.memmap module**class Memmap(context, config_path, progress_callback=None)**

Bases: `PluginInterface`

Prints the memory map

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property `open`

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects `self.validate` to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

`version = (0, 0, 0)`

volatility3.plugins.windows.mftscan module**class** **ADS**(*context*, *config_path*, *progress_callback*=None)Bases: *PluginInterface*

Scans for Alternate Data Stream

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (Optional[Callable[[float, str], None]]) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

class MFTScan(context, config_path, progress_callback=None)

Bases: *PluginInterface, TimeLinerInterface*

Scans for MFT FILE objects present in a particular windows memory image.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

generate_timeline()

Method generates Tuples of (description, timestamp_type, timestamp)

These need not be generated in any particular order, sorting will be done later

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```

unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))

```

Return type

`Dict[str, RequirementInterface]`

`version = (0, 0, 0)`

volatility3.plugins.windows.modscan module

class `ModScan(context, config_path, progress_callback=None)`

Bases: `PluginInterface`

Scans for modules present in a particular windows memory image.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod find_session_layer(context, session_layers, base_address)

Given a base address and a list of layer names, find a layer that can access the specified address.

Parameters

- **context** (`ContextInterface`) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** – The name of the layer on which to operate
- **symbol_table** – The name of the table containing the kernel symbols
- **session_layers** (`Iterable[str]`) – A list of session layer names
- **base_address** (`int`) – The base address to identify the layers that can access it

Returns

Layer name or None if no layers that contain the base address can be found

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod get_session_layers(context, layer_name, symbol_table, pids=None)

Build a cache of possible virtual layers, in priority starting with the primary/kernel layer. Then keep one layer per session by cycling through the process list.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols
- **pids** (*List[int]*) – A list of process identifiers to include exclusively or None for no filter

Return type

Generator[str, None, None]

Returns

A list of session layer names

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

classmethod `scan_modules(context, layer_name, symbol_table)`

Scans for modules using the poolscanner module and constraints.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols

Return type

Iterable[ObjectInterface]

Returns

A list of Driver objects as found from the *layer_name* layer based on Driver pool signatures

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 0)

volatility3.plugins.windows.modules module

class `Modules(context, config_path, progress_callback=None)`

Bases: *PluginInterface*

Lists the loaded kernel modules.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod find_session_layer(*context, session_layers, base_address*)

Given a base address and a list of layer names, find a layer that can access the specified address.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** – The name of the layer on which to operate
- **symbol_table** – The name of the table containing the kernel symbols
- **session_layers** (*Iterable[str]*) – A list of session layer names
- **base_address** (*int*) – The base address to identify the layers that can access it

Returns

Layer name or None if no layers that contain the base address can be found

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type*List[RequirementInterface]***classmethod get_session_layers**(*context, layer_name, symbol_table, pids=None*)

Build a cache of possible virtual layers, in priority starting with the primary/kernel layer. Then keep one layer per session by cycling through the process list.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols
- **pids** (*List[int]*) – A list of process identifiers to include exclusively or None for no filter

Return type*Generator[str, None, None]***Returns**

A list of session layer names

classmethod list_modules(*context, layer_name, symbol_table*)

Lists all the modules in the primary layer.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols

Return type*Iterable[ObjectInterface]***Returns**

A list of Modules as retrieved from PsLoadedModuleList

classmethod **make_subconfig**(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property** **open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type*None***classmethod** **unsatisfied**(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```

unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))

```

Return type`Dict[str, RequirementInterface]``version = (1, 1, 0)`**volatility3.plugins.windows.mutantscan module****class** `MutantScan(context, config_path, progress_callback=None)`Bases: `PluginInterface`

Scans for mutexes present in a particular windows memory image.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too.

Return type`HierarchicalDict`**property config:** `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

classmethod scan_mutants(*context, layer_name, symbol_table*)

Scans for mutants using the poolscanner module and constraints.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols

Return type

Iterable[ObjectInterface]

Returns

A list of Mutant objects found by scanning memory for the Mutant pool signatures

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.windows.netscan module**class NetScan**(*context, config_path, progress_callback=None*)Bases: *PluginInterface, TimeLinerInterface*

Scans for network objects present in a particular windows memory image.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

static create_netscan_constraints(*context, symbol_table*)

Creates a list of Pool Tag Constraints for network objects.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **symbol_table** (*str*) – The name of an existing symbol table containing the symbols / types

Return type*List[PoolConstraint]***Returns**

The list containing the built constraints.

classmethod create_netscan_symbol_table(*context, layer_name, nt_symbol_table, config_path*)

Creates a symbol table for TCP Listeners and TCP/UDP Endpoints.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **nt_symbol_table** (*str*) – The name of the table containing the kernel symbols

- **config_path** (*str*) – The config path where to find symbol files

Return type*str***Returns**

The name of the constructed symbol table

classmethod **determine_tcpip_version**(*context, layer_name, nt_symbol_table*)

Tries to determine which symbol filename to use for the image's tcpip driver. The logic is partially taken from the info plugin.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **nt_symbol_table** (*str*) – The name of the table containing the kernel symbols

Return type*Tuple[str, Type]***Returns**

The filename of the symbol table to use.

generate_timeline()

Method generates Tuples of (description, timestamp_type, timestamp)

These need not be generated in any particular order, sorting will be done later

classmethod **get_requirements**()

Returns a list of Requirement objects for this plugin.

classmethod **make_subconfig**(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property** **open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

classmethod `scan(context, layer_name, nt_symbol_table, netscan_symbol_table)`

Scans for network objects using the poolscanner module and constraints.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **nt_symbol_table** (*str*) – The name of the table containing the kernel symbols
- **netscan_symbol_table** (*str*) – The name of the table containing the network object symbols (`_TCP_LISTENER` etc.)

Return type

Iterable[ObjectInterface]

Returns

A list of network objects found by scanning the *layer_name* layer for network pool signatures

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 0)

volatility3.plugins.windows.netstat module

class `NetStat(context, config_path, progress_callback=None)`

Bases: *PluginInterface, TimeLinerInterface*

Traverses network tracking structures present in a particular windows memory image.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod create_tcpip_symbol_table(*context, config_path, layer_name, tcpip_module_offset, tcpip_module_size*)

DEPRECATED: Use PDBUtility.symbol_table_from_pdb instead

Creates symbol table for the current image's tcpip.sys driver.

Searches the memory section of the loaded tcpip.sys module for its PDB GUID and loads the associated symbol table into the symbol space.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **config_path** (*str*) – The config path where to find symbol files
- **layer_name** (*str*) – The name of the layer on which to operate
- **tcpip_module_offset** (*int*) – This memory dump's tcpip.sys image offset
- **tcpip_module_size** (*int*) – The size of *tcpip.sys* for this dump

Return type

str

Returns

The name of the constructed and loaded symbol table

classmethod enumerate_structures_by_port(*context, layer_name, net_symbol_table, port, port_pool_addr, proto='tcp'*)

Lists all UDP Endpoints and TCP Listeners by parsing UdpPortPool and TcpPortPool.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **net_symbol_table** (*str*) – The name of the table containing the tcpip types
- **port** (*int*) – Current port as integer to lookup the associated object.
- **port_pool_addr** (*int*) – Address of port pool object

- **proto** – Either “tcp” or “udp” to decide which types to use.

Return type`Iterable[ObjectInterface]`**Returns**

The list of network objects from this image’s TCP and UDP *PortPools*

classmethod `find_port_pools(context, layer_name, net_symbol_table, tcpip_symbol_table, tcpip_module_offset)`

Finds the given image’s port pools. Older Windows versions (presumably < Win10 build 14251) use driver symbols called *UdpPortPool* and *TcpPortPool* which point towards the pools. Newer Windows versions use *UdpCompartmentSet* and *TcpCompartmentSet*, which we first have to translate into the port pool address. See also: <http://redplait.blogspot.com/2016/06/tcpip-port-pools-in-fresh-windows-10.html>

Parameters

- **context** (`ContextInterface`) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (`str`) – The name of the layer on which to operate
- **net_symbol_table** (`str`) – The name of the table containing the tcpip types
- **tcpip_module_offset** (`int`) – This memory dump’s tcpip.sys image offset
- **tcpip_symbol_table** (`str`) – The name of the table containing the tcpip driver symbols

Return type`Tuple[int, int]`**Returns**

The tuple containing the address of the UDP and TCP port pool respectively.

generate_timeline()

Method generates Tuples of (description, timestamp_type, timestamp)

These need not be generated in any particular order, sorting will be done later

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

classmethod `get_tcpip_module(context, layer_name, nt_symbols)`

Uses *windows.modules* to find tcpip.sys in memory.

Parameters

- **context** (`ContextInterface`) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (`str`) – The name of the layer on which to operate
- **nt_symbols** (`str`) – The name of the table containing the kernel symbols

Return type`Optional[ObjectInterface]`**Returns**

The constructed tcpip.sys module object.

classmethod `list_sockets(context, layer_name, nt_symbols, net_symbol_table, tcpip_module_offset, tcpip_symbol_table)`

Lists all UDP Endpoints, TCP Listeners and TCP Endpoints in the primary layer that are in tcpip.sys’s *UdpPortPool*, *TcpPortPool* and TCP Endpoint partition table, respectively.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **nt_symbols** (*str*) – The name of the table containing the kernel symbols
- **net_symbol_table** (*str*) – The name of the table containing the tcpip types
- **tcpip_module_offset** (*int*) – Offset of *tcpip.sys*'s PE image in memory
- **tcpip_symbol_table** (*str*) – The name of the table containing the tcpip driver symbols

Return type*Iterable[ObjectInterface]***Returns**The list of network objects from the *layer_name* layer's *PartitionTable* and *PortPools***classmethod** **make_subconfig**(*context*, *base_config_path*, ***kwargs*)Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from *kwargs*.**Parameters**

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type*str***property** **open**Returns a context manager and thus can be called like *open***classmethod** **parse_bitmap**(*context*, *layer_name*, *bitmap_offset*, *bitmap_size_in_byte*)

Parses a given bitmap and looks for each occurrence of a 1.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **bitmap_offset** (*int*) – Start address of bitmap
- **bitmap_size_in_byte** (*int*) – Bitmap size in Byte, not in bit.

Return type*list***Returns**

The list of indices at which a 1 was found.

classmethod `parse_hashtable(context, layer_name, ht_offset, ht_length, alignment, net_symbol_table)`

Parses a hashtable quick and dirty.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **ht_offset** (*int*) – Beginning of the hash table
- **ht_length** (*int*) – Length of the hash table

Return type

Generator[*ObjectInterface*, *None*, *None*]

Returns

The hash table entries which are `_not_` empty

classmethod `parse_partitions(context, layer_name, net_symbol_table, tcpip_symbol_table, tcpip_module_offset)`

Parses tcpip.sys's PartitionTable containing established TCP connections. The amount of Partition depends on the value of the symbol *PartitionCount* and correlates with the maximum processor count (refer to Art of Memory Forensics, chapter 11).

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **net_symbol_table** (*str*) – The name of the table containing the tcpip types
- **tcpip_symbol_table** (*str*) – The name of the table containing the tcpip driver symbols
- **tcpip_module_offset** (*int*) – The offset of the tcpip module

Return type

Iterable[*ObjectInterface*]

Returns

The list of TCP endpoint objects from the *layer_name* layer's *PartitionTable*

classmethod `read_pointer(context, layer_name, offset, length)`

Reads a pointer at a given offset and returns the address it points to.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **offset** (*int*) – Offset of pointer
- **length** (*int*) – Pointer length

Return type

int

Returns

The value the pointer points to.

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 0)

volatility3.plugins.windows.poolscanner module

class PoolConstraint(tag, type_name, object_type=None, page_type=None, size=None, index=None, alignment=1, skip_type_test=False, additional_structures=None)

Bases: `object`

Class to maintain tag/size/index/type information about Pool header tags.

class PoolHeaderScanner(module, constraint_lookup, alignment)

Bases: `ScannerInterface`

property context: `ContextInterface` | `None`

property layer_name: `str` | `None`

thread_safe = False

version = (0, 0, 0)

class PoolScanner(context, config_path, progress_callback=None)

Bases: `PluginInterface`

A generic pool scanner plugin.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

static builtin_constraints(symbol_table, tags_filter=None)

Get built-in PoolConstraints given a list of pool tags.

The tags_filter is a list of pool tags, and the associated PoolConstraints are returned. If tags_filter is empty or not supplied, then all builtin constraints are returned.

Parameters

- **symbol_table** (*str*) – The name of the symbol table to prepend to the types used
- **tags_filter** (*List[bytes]*) – List of tags to return or None to return all

Return type

List[PoolConstraint]

Returns

A list of well-known constructed PoolConstraints that match the provided tags

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod generate_pool_scan(context, layer_name, symbol_table, constraints)**Parameters**

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols
- **constraints** (*List[PoolConstraint]*) – List of pool constraints used to limit the scan results

Return type

Generator[Tuple[PoolConstraint, ObjectInterface, ObjectInterface], None, None]

Returns

Iterable of tuples, containing the constraint that matched, the object from memory, the object header used to determine the object

classmethod `get_pool_header_table(context, symbol_table)`

Returns the appropriate symbol_table containing a _POOL_HEADER type, even if the original symbol table doesn't contain one.

Parameters

- **context** (*ContextInterface*) – The context that the symbol tables does (or will) reside in
- **symbol_table** (*str*) – The expected symbol_table to contain the _POOL_HEADER type

Return type

str

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

classmethod `pool_scan(context, layer_name, symbol_table, pool_constraints, alignment=8, progress_callback=None)`

Returns the _POOL_HEADER object (based on the symbol_table template) after scanning through layer_name returning all headers that match any of the constraints provided. Only one constraint can be provided per tag.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols
- **pool_constraints** (*List[PoolConstraint]*) – List of pool constraints used to limit the scan results

- **alignment** (`int`) – An optional value that all pool headers will be aligned to
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – An optional function to provide progress feedback whilst scanning

Return type`Generator[Tuple[PoolConstraint, ObjectInterface], None, None]`**Returns**

An Iterable of pool constraints and the pool headers associated with them

run()

Executes the functionality of the code.

Note: This method expects `self.validate` to have been called to ensure all necessary options have been provided

Return type`TreeGrid`**Returns**

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type`None`**classmethod unsatisfied(context, config_path)**

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`

version = (1, 0, 0)

class PoolType(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)

Bases: `IntFlag`

Class to maintain the different possible PoolTypes The values must be integer powers of 2.

FREE = 4

NONPAGED = 2

PAGED = 1

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If `byteorder` is `'big'`, the most significant byte is at the beginning of the byte array. If `byteorder` is `'little'`, the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `'sys.byteorder'` as the byte order value. Default is to use `'big'`.

signed

Determines whether two's complement is used to represent the integer. If `signed` is `False` and a negative integer is given, an `OverflowError` is raised.

volatility3.plugins.windows.privileges module**class Privs**(*args, **kwargs)

Bases: [*PluginInterface*](#)

Lists process token privileges

Parameters

- **context** – The context that the plugin will operate within
- **config_path** – The path to configuration data within the context configuration data
- **progress_callback** – A callable that can provide feedback at progress points

build_configuration()

Constructs a `HierarchicalDictionary` of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too.

Return type

[*HierarchicalDict*](#)

property config: [*HierarchicalDict*](#)

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: [*ContextInterface*](#)

The context object that this configurable belongs to/configuration is stored in.

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property `open`

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects `self.validate` to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (1, 2, 0)

volatility3.plugins.windows.pslist module**class PsList**(*context, config_path, progress_callback=None*)Bases: *PluginInterface, TimeLinerInterface*

Lists the processes present in a particular windows memory image.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

PHYSICAL_DEFAULT = False**build_configuration()**

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod create_name_filter(*name_list=None, exclude=False*)

A factory for producing filter functions that filter based on a list of process names.

Parameters

- **name_list** (*List[str]*) – A list of process names that are acceptable, all other processes will be filtered out
- **exclude** (*bool*) – Accept only tasks that are not in name_list

Return type*Callable[[ObjectInterface], bool]***Returns**Filter function for passing to the *list_processes* method**classmethod create_pid_filter**(*pid_list=None, exclude=False*)

A factory for producing filter functions that filter based on a list of process IDs.

Parameters

- **pid_list** (*List[int]*) – A list of process IDs that are acceptable, all other processes will be filtered out
- **exclude** (*bool*) – Accept only tasks that are not in pid_list

Return type`Callable[[ObjectInterface], bool]`**Returns**Filter function for passing to the *list_processes* method**generate_timeline()**

Method generates Tuples of (description, timestamp_type, timestamp)

These need not be generated in any particular order, sorting will be done later

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod list_processes(context, layer_name, symbol_table, filter_func=<function PsList.<lambda>>)

Lists all the processes in the primary layer that are in the pid config option.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols
- **filter_func** (*Callable[[ObjectInterface], bool]*) – A function which takes an EPROCESS object and returns True if the process should be ignored/filtered

Return type`Iterable[ObjectInterface]`**Returns**The list of EPROCESS objects from the *layer_name* layer's PsActiveProcessHead list after filtering**classmethod make_subconfig(context, base_config_path, **kwargs)**

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property open**

Returns a context manager and thus can be called like open

classmethod process_dump(context, kernel_table_name, pe_table_name, proc, open_method)

Extracts the complete data for a process as a FileHandlerInterface

Parameters

- **context** (*ContextInterface*) – the context to operate upon
- **kernel_table_name** (*str*) – the name for the symbol table containing the kernel’s symbols
- **pe_table_name** (*str*) – the name for the symbol table containing the PE format symbols
- **proc** (*ObjectInterface*) – the process object whose memory should be output
- **open_method** (*Type[FileHandlerInterface]*) – class to provide context manager for opening the file

Return type*FileHandlerInterface***Returns**

An open FileHandlerInterface object containing the complete data for the process or None in the case of failure

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type*None***classmethod unsatisfied(context, config_path)**

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type*Dict[str, RequirementInterface]*

version = (2, 0, 0)

volatility3.plugins.windows.psscan module

class `PsScan(context, config_path, progress_callback=None)`

Bases: `PluginInterface`, `TimeLinerInterface`

Scans for processes present in a particular windows memory image.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

generate_timeline()

Method generates Tuples of (description, timestamp_type, timestamp)

These need not be generated in any particular order, sorting will be done later

classmethod get_osversion(context, layer_name, symbol_table)

Returns the complete OS version (MAJ,MIN,BUILD)

Parameters

- **context** (`ContextInterface`) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (`str`) – The name of the layer on which to operate
- **symbol_table** (`str`) – The name of the table containing the kernel symbols

Return type

`Tuple[int, int, int]`

Returns

A tuple with (MAJ,MIN,BUILD)

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

classmethod `scan_processes(context, layer_name, symbol_table, filter_func=<function PsScan.<lambda>>)`

Scans for processes using the poolscanner module and constraints.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols

Return type

Iterable[ObjectInterface]

Returns

A list of processes found by scanning the *layer_name* layer for process pool signatures

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```

unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))

```

Return type`Dict[str, RequirementInterface]``version = (1, 1, 0)`**classmethod** `virtual_process_from_physical(context, layer_name, symbol_table, proc)`

Returns a virtual process from a physical addressed one

Parameters

- **context** (`ContextInterface`) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (`str`) – The name of the layer on which to operate
- **symbol_table** (`str`) – The name of the table containing the kernel symbols
- **proc** (`ObjectInterface`) – the process object with physical address

Return type`Optional[ObjectInterface]`**Returns**

A process object on virtual address layer

volatility3.plugins.windows.pstree module**class** `PsTree(*args, **kwargs)`Bases: `PluginInterface`

Plugin for listing processes in a tree based on their parent process ID.

Parameters

- **context** – The context that the plugin will operate within
- **config_path** – The path to configuration data within the context configuration data
- **progress_callback** – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type`HierarchicalDict`**property config:** `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: [`ContextInterface`](#)

The context object that this configurable belongs to/configuration is stored in.

find_level(*pid*, *filter_func*=<function *PsTree*.<lambda>>)

Finds how deep the pid is in the processes list.

Return type

[`None`](#)

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** ([`ContextInterface`](#)) – The context in which to store the new configuration
- **base_config_path** ([`str`](#)) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

[`str`](#)

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

[`None`](#)

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```


Return type`Dict[str, RequirementInterface]``version = (0, 0, 0)`**volatility3.plugins.windows.sessions module****class** `Sessions(context, config_path, progress_callback=None)`Bases: `PluginInterface`, `TimeLinerInterface`

lists Processes with Session information extracted from Environmental Variables

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type`HierarchicalDict`**property config:** `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

generate_timeline()

Method generates Tuples of (description, timestamp_type, timestamp)

These need not be generated in any particular order, sorting will be done later

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(`context, base_config_path, **kwargs`)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.plugins.windows.skeleton_key_check module

class Skeleton_Key_Check(context, config_path, progress_callback=None)

Bases: *PluginInterface*

Looks for signs of Skeleton Key malware

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional*[Callable[[float, str], None]]) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.plugins.windows.ssd module

class `SSDT(context, config_path, progress_callback=None)`

Bases: `PluginInterface`

Lists the system call table.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

classmethod `build_module_collection(context, layer_name, symbol_table)`

Builds a collection of modules.

Parameters

- **context** (`ContextInterface`) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (`str`) – The name of the layer on which to operate
- **symbol_table** (`str`) – The name of the table containing the kernel symbols

Return type

`ModuleCollection`

Returns

A Module collection of available modules based on `Modules.list_modules`

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Return type

TreeGrid

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

`version = (1, 0, 0)`

volatility3.plugins.windows.strings module

class `Strings(context, config_path, progress_callback=None)`

Bases: `PluginInterface`

Reads output from the strings command and indicates which process(es) each string belongs to.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod generate_mapping(context, layer_name, symbol_table, progress_callback=None, pid_list=None)

Creates a reverse mapping between virtual addresses and physical addresses.

Parameters

- **context** (`ContextInterface`) – the context for the method to run against
- **layer_name** (`str`) – the layer to map against the string lines
- **symbol_table** (`str`) – the name of the symbol table for the provided layer
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – an optional callable to display progress

- **pid_list** (`Optional[List[int]]`) – a list of process IDs to consider when generating the reverse map

Return type`Dict[int, Set[Tuple[str, int]]]`**Returns**

A mapping of virtual offsets to strings and physical offsets

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type`List[RequirementInterface]`**classmethod make_subconfig(context, base_config_path, **kwargs)**

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects `self.validate` to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type`None``strings_pattern = re.compile(b'^(?:\\W*)([0-9]+)(?:\\W*)(\\w[\\w\\W]+)\\n?')`**classmethod unsatisfied(context, config_path)**

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

`version = (1, 2, 0)`

volatility3.plugins.windows.svcscan module

class `SvcScan(context, config_path, progress_callback=None)`

Bases: `PluginInterface`

Scans for windows services.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

static create_service_table(context, symbol_table, config_path)

Constructs a symbol table containing the symbols for services depending upon the operating system in use.

Parameters

- **context** (`ContextInterface`) – The context to retrieve required elements (layers, symbol tables) from
- **symbol_table** (`str`) – The name of the table containing the kernel symbols
- **config_path** (`str`) – The configuration path for any settings required by the new table

Return type

`str`

Returns

A symbol table containing the symbols necessary for services

static `get_record_tuple(service_record)`

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property `open`

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects `self.validate` to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]``version = (1, 0, 0)`**volatility3.plugins.windows.symlinkscan module****class** `SymlinkScan(context, config_path, progress_callback=None)`Bases: `PluginInterface`, `TimeLinerInterface`

Scans for links present in a particular windows memory image.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too.

Return type`HierarchicalDict`**property config:** `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

generate_timeline()

Method generates Tuples of (description, timestamp_type, timestamp)

These need not be generated in any particular order, sorting will be done later

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

classmethod make_subconfig(`context, base_config_path, **kwargs`)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

classmethod scan_symlinks(context, layer_name, symbol_table)

Scans for links using the poolscanner module and constraints.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (`str`) – The name of the layer on which to operate
- **symbol_table** (`str`) – The name of the table containing the kernel symbols

Return type

Iterable[ObjectInterface]

Returns

A list of symlink objects found by scanning memory for the Symlink pool signatures

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.windows.vadinfo module**class VadInfo(*args, **kwargs)**Bases: *PluginInterface*

Lists process memory ranges.

Parameters

- **context** – The context that the plugin will operate within
- **config_path** – The path to configuration data within the context configuration data
- **progress_callback** – A callable that can provide feedback at progress points

MAXSIZE_DEFAULT = 1073741824**build_configuration()**

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type*List[RequirementInterface]***classmethod list_vads(proc, filter_func=<function VadInfo.<lambda>>)**

Lists the Virtual Address Descriptors of a specific process.

Parameters

- **proc** (*ObjectInterface*) – _EPROCESS object from which to list the VADs
- **filter_func** (*Callable[[ObjectInterface], bool]*) – Function to take a virtual address descriptor value and return True if it should be filtered out

Return type*Generator[ObjectInterface, None, None]***Returns**

A list of virtual address descriptors based on the process and filtered based on the filter function

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

classmethod protect_values(*context, layer_name, symbol_table*)

Look up the array of memory protection constants from the memory sample. These don't change often, but if they do in the future, then finding them dynamically versus hard-coding here will ensure we parse them properly.

Parameters

- **context** (*ContextInterface*) – The context to retrieve required elements (layers, symbol tables) from
- **layer_name** (*str*) – The name of the layer on which to operate
- **symbol_table** (*str*) – The name of the table containing the kernel symbols

Return type

Iterable[int]

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(*context, config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`**classmethod** `vad_dump(context, proc, vad, open_method, maxsize=1073741824)`

Extracts the complete data for Vad as a FileInterface.

Parameters

- **context** (`ContextInterface`) – The context to retrieve required elements (layers, symbol tables) from
- **proc** (`ObjectInterface`) – an `_EPROCESS` instance
- **vad** (`ObjectInterface`) – The suspected VAD to extract (`ObjectInterface`)
- **open_method** (`Type[FileHandlerInterface]`) – class to provide context manager for opening the file
- **maxsize** (`int`) – Max size of VAD section (default `MAXSIZE_DEFAULT`)

Return type`Optional[FileHandlerInterface]`**Returns**

An open FileInterface object containing the complete data for the process or None in the case of failure

version = (2, 0, 0)**volatility3.plugins.windows.vadwalk module****class** `VadWalk(context, config_path, progress_callback=None)`Bases: `PluginInterface`

Walk the VAD tree.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type`HierarchicalDict`**property config:** `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod make_subconfig(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Return type

`TreeGrid`

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(*handler*)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod unsatisfied(*context*, *config_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]``version = (1, 0, 0)`**volatility3.plugins.windows.vadparascan module****class** `VadYaraScan(context, config_path, progress_callback=None)`Bases: `PluginInterface`

Scans all the Virtual Address Descriptor memory maps using yara.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too.

Return type`HierarchicalDict`**property config:** `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type`List[RequirementInterface]`**static get_vad_maps(task)**

Creates a map of start/end addresses within a virtual address descriptor tree.

Parameters**task** (`ObjectInterface`) – The EPROCESS object of which to traverse the vad tree**Return type**`Iterable[Tuple[int, int]]`**Returns**

An iterable of tuples containing start and end addresses for each descriptor

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 1)

volatility3.plugins.windows.verinfo module**class VerInfo**(*context, config_path, progress_callback=None*)Bases: *PluginInterface*

Lists version information from PE files.

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type*HierarchicalDict***property config:** *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod find_version_info(*context, layer_name, filename*)

Searches for an original filename, then tracks back to find the VS_VERSION_INFO and read the fixed version information structure

Return type*Optional[Tuple[int, int, int, int]]***classmethod get_requirements()**

Returns a list of Requirement objects for this plugin.

Return type*List[RequirementInterface]***classmethod get_version_information**(*context, pe_table_name, layer_name, base_address*)

Get File and Product version information from PE files.

Parameters

- **context** (*ContextInterface*) – volatility context on which to operate
- **pe_table_name** (*str*) – name of the PE table
- **layer_name** (*str*) – name of the layer containing the PE file
- **base_address** (*int*) – base address of the PE (where MZ is found)

Return type*Tuple[int, int, int, int]*

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (1, 0, 0)

volatility3.plugins.windows.virtmap module**class** `VirtMap`(*context*, *config_path*, *progress_callback*=None)Bases: `PluginInterface`

Lists virtual mapped sections.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type`HierarchicalDict`**property config:** `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod determine_map(*module*)

Returns the virtual map from a windows kernel module.

Return type`Dict[str, List[Tuple[int, int]]]`**classmethod get_requirements()**

Returns a list of Requirement objects for this plugin.

Return type`List[RequirementInterface]`**classmethod make_subconfig**(*context*, *base_config_path*, ***kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property open**

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

classmethod scannable_sections(module)**Return type**`Generator[Tuple[int, int], None, None]`**set_open_method(handler)**

Sets the file handler to be used by this plugin.

Return type`None`**classmethod unsatisfied(context, config_path)**

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type`Dict[str, RequirementInterface]`

version = (0, 0, 0)

Submodules

volatility3.plugins.banners module

class Banners(context, config_path, progress_callback=None)

Bases: `PluginInterface`

Attempts to identify potential linux banners in an image

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data

- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod locate_banners(*context, layer_name*)

Identifies banners from a memory image

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.plugins.configwriter module**class ConfigWriter(context, config_path, progress_callback=None)**

Bases: `PluginInterface`

Runs the automagics and both prints and outputs configuration in the output directory.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: HierarchicalDict

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: str

The configuration path on which this configurable lives.

property context: ContextInterface

The context object that this configurable belongs to/configuration is stored in.

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property `open`

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects `self.validate` to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.plugins.frameworkinfo module

class FrameworkInfo(*context, config_path, progress_callback=None*)

Bases: *PluginInterface*

Plugin to list the various modular components of Volatility

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod unsatisfied(context, config_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (0, 0, 0)

volatility3.plugins.isfinfo module**class IsfInfo(context, config_path, progress_callback=None)**

Bases: *PluginInterface*

Determines information about the currently available ISF files, or a specific one

Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config_path** (*str*) – The path to configuration data within the context configuration data
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

HierarchicalDict

property config: *HierarchicalDict*

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: *str*

The configuration path on which this configurable lives.

property context: *ContextInterface*

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

List[RequirementInterface]

classmethod list_all_isf_files()

Lists all the ISF files that can be found

Return type

Generator[str, None, None]

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (2, 0, 0)

volatility3.plugins.layerwriter module

class `LayerWriter(context, config_path, progress_callback=None)`

Bases: `PluginInterface`

Runs the automagics and writes out the primary layer produced by the stacker.

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built. Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type

`HierarchicalDict`

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

default_block_size = 5242880

classmethod `get_requirements()`

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod `make_subconfig(context, base_config_path, **kwargs)`

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base_config_path** (*str*) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

str

property open

Returns a context manager and thus can be called like open

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

None

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

Dict[str, RequirementInterface]

version = (2, 0, 0)

classmethod `write_layer(context, layer_name, preferred_name, open_method, chunk_size=None, progress_callback=None)`

Produces a FileHandler from the named layer in the provided context or None on failure

Parameters

- **context** (*ContextInterface*) – the context from which to read the memory layer
- **layer_name** (*str*) – the name of the layer to write out
- **preferred_name** (*str*) – a string with the preferred filename for hte file
- **chunk_size** (*Optional[int]*) – an optional size for the chunks that should be written (defaults to 0x500000)
- **open_method** (*Type[FileHandlerInterface]*) – class for creating FileHandler context managers
- **progress_callback** (*Optional[Callable[[float, str], None]]*) – an optional function that takes a percentage and a string that displays output

Return type*Optional[FileHandlerInterface]***volatility3.plugins.timeliner module****class TimeLinerInterface**Bases: *object*

Interface defining methods that timeliner will use to generate a body file.

abstract generate_timeline()

Method generates Tuples of (description, timestamp_type, timestamp)

These need not be generated in any particular order, sorting will be done later

Return type*Generator[Tuple[str, TimeLinerType, datetime], None, None]***class TimeLinerType**(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)Bases: *IntEnum***ACCESSED** = 3**CHANGED** = 4**CREATED** = 1**MODIFIED** = 2**as_integer_ratio()**

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An OverflowError is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

class `Timeliner(*args, **kwargs)`

Bases: `PluginInterface`

Runs all relevant plugins that provide time related information and orders the results by time.

Parameters

- **context** – The context that the plugin will operate within
- **config_path** – The path to configuration data within the context configuration data
- **progress_callback** – A callable that can provide feedback at progress points

build_configuration()

Builds the configuration to save for the plugin such that it can be reconstructed.

property config: `HierarchicalDict`

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns a list of Requirement objects for this plugin.

Return type

`List[RequirementInterface]`

classmethod get_usable_plugins(*selected_list=None*)

Return type

`List[Type]`

classmethod make_subconfig(*context, base_config_path, **kwargs*)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type`str`**property open**

Returns a context manager and thus can be called like open

run()

Isolate each plugin and run it.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type`None`**classmethod unsatisfied(context, config_path)**

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```

unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))

```

Return type`Dict[str, RequirementInterface]`

version = (0, 0, 0)

volatility3.plugins.yarascan module**class YaraScan(context, config_path, progress_callback=None)**

Bases: `PluginInterface`

Scans kernel memory using yara rules (string or file).

Parameters

- **context** (`ContextInterface`) – The context that the plugin will operate within
- **config_path** (`str`) – The path to configuration data within the context configuration data
- **progress_callback** (`Optional[Callable[[float, str], None]]`) – A callable that can provide feedback at progress points

build_configuration()

Constructs a HierarchicalDictionary of all the options required to build this component in the current context.

Ensures that if the class has been created, it can be recreated using the configuration built Inheriting classes must override this to ensure any dependent classes update their configurations too

Return type`HierarchicalDict`**property config: HierarchicalDict**

The Hierarchical configuration Dictionary for this Configurable object.

property config_path: `str`

The configuration path on which this configurable lives.

property context: `ContextInterface`

The context object that this configurable belongs to/configuration is stored in.

classmethod get_requirements()

Returns the requirements needed to run yarascan directly, combining the TranslationLayerRequirement and the requirements from get_yarascan_option_requirements.

Return type

`List[RequirementInterface]`

classmethod get_yarascan_option_requirements()

Returns the requirements needed for the command lines options used by yarascan. This can then also be used by other plugins that are using yarascan. This does not include a TranslationLayerRequirement or a ModuleRequirement.

Return type

`List[RequirementInterface]`

classmethod make_subconfig(context, base_config_path, **kwargs)

Convenience function to allow constructing a new randomly generated sub-configuration path, containing each element from kwargs.

Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base_config_path** (`str`) – The base configuration path on which to build the new configuration
- **kwargs** – Keyword arguments that are used to populate the new configuration path

Returns

The newly generated full configuration path

Return type

`str`

property open

Returns a context manager and thus can be called like open

classmethod process_yara_options(config)

run()

Executes the functionality of the code.

Note: This method expects *self.validate* to have been called to ensure all necessary options have been provided

Returns

A TreeGrid object that can then be passed to a Renderer.

set_open_method(handler)

Sets the file handler to be used by this plugin.

Return type

`None`

classmethod `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be used in tests as follows:

```
unmet = configurable.unsatisfied(context, config_path)
if unmet:
    raise RuntimeError("Unsatisfied requirements: {}".format(unmet))
```

Return type

`Dict[str, RequirementInterface]`

version = (1, 2, 0)

class `YaraScanner(rules)`

Bases: `ScannerInterface`

property `context`: `ContextInterface` | `None`

property `layer_name`: `str` | `None`

thread_safe = `False`

version = (2, 0, 0)

10.1.4 volatility3.schemas package

create_json_hash(input, schema=None)

Constructs the hash of the input and schema to create a unique identifier for a particular JSON file.

Return type

`Optional[str]`

load_cached_validations()

Loads up the list of successfully cached json objects, so we don't need to revalidate them.

Return type

`Set[str]`

record_cached_validations(validations)

Record the cached validations, so we don't need to revalidate them in future.

Return type

`None`

valid(input, schema, use_cache=True)

Validates a json schema.

Return type

`bool`

validate(input, use_cache=True)

Validates an input JSON file based upon.

Return type

`bool`

10.1.5 volatility3.symbols package

Defines the symbols architecture.

This is the namespace for all volatility symbols, and determines the path for loading symbol ISF files

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

V

volatility3, 47
volatility3.cli, 48
volatility3.cli.text_renderer, 64
volatility3.cli.volargparse, 67
volatility3.cli.volshell, 49
volatility3.cli.volshell.generic, 50
volatility3.cli.volshell.linux, 55
volatility3.cli.volshell.mac, 58
volatility3.cli.volshell.windows, 61
volatility3.framework, 68
volatility3.framework.automagic, 69
volatility3.framework.automagic.construct_layers, 70
volatility3.framework.automagic.linux, 72
volatility3.framework.automagic.mac, 74
volatility3.framework.automagic.module, 77
volatility3.framework.automagic.pdbscan, 78
volatility3.framework.automagic.stacker, 82
volatility3.framework.automagic.symbol_cache, 85
volatility3.framework.automagic.symbol_finder, 90
volatility3.framework.automagic.windows, 92
volatility3.framework.configuration, 95
volatility3.framework.configuration.requirements, 95
volatility3.framework.constants, 119
volatility3.framework.constants.linux, 122
volatility3.framework.constants.windows, 122
volatility3.framework.contexts, 123
volatility3.framework.exceptions, 555
volatility3.framework.interfaces, 134
volatility3.framework.interfaces.automagic, 134
volatility3.framework.interfaces.configuration, 137
volatility3.framework.interfaces.context, 148
volatility3.framework.interfaces.layers, 154
volatility3.framework.interfaces.objects, 161
volatility3.framework.interfaces.plugins, 165
volatility3.framework.interfaces.renderers, 168
volatility3.framework.interfaces.symbols, 172
volatility3.framework.layers, 180
volatility3.framework.layers.avml, 182
volatility3.framework.layers.cloudstorage, 185
volatility3.framework.layers.codecs, 180
volatility3.framework.layers.crash, 185
volatility3.framework.layers.elf, 191
volatility3.framework.layers.intel, 195
volatility3.framework.layers.leechcore, 214
volatility3.framework.layers.lime, 214
volatility3.framework.layers.linear, 217
volatility3.framework.layers.msf, 220
volatility3.framework.layers.physical, 225
volatility3.framework.layers.qemu, 230
volatility3.framework.layers.registry, 233
volatility3.framework.layers.resources, 237
volatility3.framework.layers.scanners, 180
volatility3.framework.layers.scanners.multiregexp, 181
volatility3.framework.layers.segmented, 239
volatility3.framework.layers.vmware, 243
volatility3.framework.layers.xen, 246
volatility3.framework.objects, 249
volatility3.framework.objects.templates, 298
volatility3.framework.objects.utility, 300
volatility3.framework.plugins, 301
volatility3.framework.renderers, 301
volatility3.framework.renderers.conversion, 304
volatility3.framework.renderers.format_hints, 305
volatility3.framework.symbols, 316
volatility3.framework.symbols.generic, 319
volatility3.framework.symbols.intermed, 523
volatility3.framework.symbols.linux, 321
volatility3.framework.symbols.linux.bash, 397
volatility3.framework.symbols.linux.extensions, 326
volatility3.framework.symbols.linux.extensions.bash, 390

volatility3.framework.symbols.linux.extensions, 392
volatility3.framework.symbols.mac, 401
volatility3.framework.symbols.mac.extensions, 405
volatility3.framework.symbols.metadata, 553
volatility3.framework.symbols.native, 553
volatility3.framework.symbols.windows, 429
volatility3.framework.symbols.windows.extensions, 432
volatility3.framework.symbols.windows.extensions.linux, 470
volatility3.framework.symbols.windows.extensions.macos, 472
volatility3.framework.symbols.windows.extensions.macos.linux, 474
volatility3.framework.symbols.windows.extensions.macos.linux.linux, 479
volatility3.framework.symbols.windows.extensions.macos.linux.macos, 485
volatility3.framework.symbols.windows.extensions.macos.linux.macos.linux, 485
volatility3.framework.symbols.windows.extensions.macos.linux.macos.linux.linux, 489
volatility3.framework.symbols.windows.extensions.macos.linux.macos.linux.macos, 500
volatility3.framework.symbols.windows.extensions.macos.linux.macos.linux.macos.linux, 512
volatility3.framework.symbols.windows.pdbconv, 517
volatility3.framework.symbols.windows.pdbutil, 519
volatility3.framework.symbols.windows.versions, 522
volatility3.framework.symbols.wrappers, 555
volatility3.plugins, 558
volatility3.plugins.banners, 753
volatility3.plugins.configwriter, 755
volatility3.plugins.frameworkinfo, 757
volatility3.plugins.isinfo, 758
volatility3.plugins.layerwriter, 760
volatility3.plugins.linux, 558
volatility3.plugins.linux.bash, 559
volatility3.plugins.linux.capabilities, 560
volatility3.plugins.linux.check_afinfo, 563
volatility3.plugins.linux.check_creds, 565
volatility3.plugins.linux.check_idt, 566
volatility3.plugins.linux.check_modules, 568
volatility3.plugins.linux.check_syscall, 569
volatility3.plugins.linux.elfs, 571
volatility3.plugins.linux.envvars, 573
volatility3.plugins.linux.iomem, 575
volatility3.plugins.linux.keyboard_notifiers, 577
volatility3.plugins.linux.kmsg, 578
volatility3.plugins.linux.lsmod, 585
volatility3.plugins.linux.lsof, 587
volatility3.plugins.linux.malfind, 588
volatility3.plugins.linux.mountinfo, 590
volatility3.plugins.linux.proc, 592
volatility3.plugins.linux.psaux, 595
volatility3.plugins.linux.pslist, 596
volatility3.plugins.linux.psscan, 599
volatility3.plugins.linux.pstree, 601
volatility3.plugins.linux.sockstat, 602
volatility3.plugins.linux.tty_check, 605
volatility3.plugins.linux.vmayarascan, 606
volatility3.plugins.mac, 608
volatility3.plugins.mac.bash, 608
volatility3.plugins.mac.check_syscall, 610
volatility3.plugins.mac.check_sysctl, 612
volatility3.plugins.mac.check_trap_table, 613
volatility3.plugins.mac.ifconfig, 615
volatility3.plugins.mac.kauth_listeners, 616
volatility3.plugins.mac.kauth_scopes, 618
volatility3.plugins.mac.kevents, 620
volatility3.plugins.mac.list_files, 622
volatility3.plugins.mac.lsmod, 623
volatility3.plugins.mac.lsof, 625
volatility3.plugins.mac.malfind, 627
volatility3.plugins.mac.mount, 628
volatility3.plugins.mac.netstat, 630
volatility3.plugins.mac.proc_maps, 632
volatility3.plugins.mac.psaux, 633
volatility3.plugins.mac.pslist, 635
volatility3.plugins.mac.pstree, 639
volatility3.plugins.mac.socket_filters, 640
volatility3.plugins.mac.timers, 642
volatility3.plugins.mac.trustedbsd, 643
volatility3.plugins.mac.vfsevents, 645
volatility3.plugins.timeliner, 762
volatility3.plugins.windows, 647
volatility3.plugins.windows.bigpools, 655
volatility3.plugins.windows.cachedump, 657
volatility3.plugins.windows.callbacks, 659
volatility3.plugins.windows.cmdline, 662
volatility3.plugins.windows.crashinfo, 664
volatility3.plugins.windows.devicetree, 666
volatility3.plugins.windows.dlllist, 667
volatility3.plugins.windows.driverirp, 669
volatility3.plugins.windows.drivermodule, 671
volatility3.plugins.windows.driverscan, 673
volatility3.plugins.windows.dumpfiles, 675
volatility3.plugins.windows.envvars, 677
volatility3.plugins.windows.filescan, 678
volatility3.plugins.windows.getservicesids, 680
volatility3.plugins.windows.getsids, 682

- volatility3.plugins.windows.handles, 684
- volatility3.plugins.windows.hashdump, 686
- volatility3.plugins.windows.info, 689
- volatility3.plugins.windows.joblinks, 691
- volatility3.plugins.windows.ldrmodules, 693
- volatility3.plugins.windows.lsadump, 694
- volatility3.plugins.windows.malfind, 696
- volatility3.plugins.windows.mbrscan, 698
- volatility3.plugins.windows.memmap, 700
- volatility3.plugins.windows.mftscan, 702
- volatility3.plugins.windows.modscan, 705
- volatility3.plugins.windows.modules, 707
- volatility3.plugins.windows.mutantscan, 710
- volatility3.plugins.windows.netscan, 712
- volatility3.plugins.windows.netstat, 714
- volatility3.plugins.windows.poolscanner, 719
- volatility3.plugins.windows.privileges, 724
- volatility3.plugins.windows.pslist, 726
- volatility3.plugins.windows.psscan, 729
- volatility3.plugins.windows.pstree, 731
- volatility3.plugins.windows.registry, 647
- volatility3.plugins.windows.registry.hivelist, 647
- volatility3.plugins.windows.registry.hivescan, 650
- volatility3.plugins.windows.registry.printkey, 652
- volatility3.plugins.windows.registry.userassist, 654
- volatility3.plugins.windows.sessions, 733
- volatility3.plugins.windows.skeleton_key_check, 734
- volatility3.plugins.windows.ssd, 736
- volatility3.plugins.windows.strings, 738
- volatility3.plugins.windows.svcscan, 740
- volatility3.plugins.windows.symblinkscan, 742
- volatility3.plugins.windows.vadinfo, 744
- volatility3.plugins.windows.vadwalk, 746
- volatility3.plugins.windows.vadyarascan, 748
- volatility3.plugins.windows.verinfo, 750
- volatility3.plugins.windows.virtmap, 752
- volatility3.plugins.yarascan, 765
- volatility3.schemas, 767
- volatility3.symbols, 768

A

- ABCKmsg (class in volatility3.plugins.linux.kmsg), 578
- access_string() (FILE_OBJECT method), 446
- ACCESSED (TimeLinerType attribute), 762
- add_argument() (HelpfulArgParser method), 67
- add_argument_group() (HelpfulArgParser method), 67
- add_identifier() (CacheManagerInterface method), 85
- add_identifier() (SqliteCache method), 87
- add_layer() (Context method), 126
- add_layer() (ContextInterface method), 148
- add_layer() (LayerContainer method), 156
- add_module() (Context method), 126
- add_module() (ContextInterface method), 149
- add_module() (ModuleCollection method), 130
- add_module() (ModuleContainer method), 150
- add_mutually_exclusive_group() (HelpfulArgParser method), 67
- add_note() (ElfFormatException method), 194
- add_note() (InvalidAddressException method), 555
- add_note() (LayerException method), 556
- add_note() (LimeFormatException method), 214
- add_note() (MissingModuleException method), 556
- add_note() (OfflineException method), 556
- add_note() (PagedInvalidAddressException method), 556
- add_note() (PDBFormatException method), 220
- add_note() (PluginRequirementException method), 557
- add_note() (PluginVersionException method), 557
- add_note() (RegistryFormatException method), 233
- add_note() (RegistryInvalidIndex method), 237
- add_note() (SnappyException method), 184
- add_note() (SwappedInvalidAddressException method), 557
- add_note() (SymbolError method), 557
- add_note() (SymbolSpaceError method), 557
- add_note() (UnsatisfiedException method), 558
- add_note() (VmwareFormatException method), 243
- add_note() (VolatilityException method), 558
- add_note() (WindowsCrashDumpFormatException method), 190
- add_parent() (JarHandler method), 237
- add_parent() (OfflineHandler method), 237
- add_parent() (VolatilityHandler method), 238
- add_parser() (HelpfulSubparserAction method), 68
- add_pattern() (MultiRegexp method), 181
- add_process_layer() (EPROCESS method), 439
- add_process_layer() (proc method), 413
- add_process_layer() (task_struct method), 377
- add_requirement() (BooleanRequirement method), 96
- add_requirement() (BytesRequirement method), 97
- add_requirement() (ChoiceRequirement method), 98
- add_requirement() (ClassRequirement method), 137
- add_requirement() (ComplexListRequirement method), 99
- add_requirement() (ConfigurableRequirementInterface method), 140
- add_requirement() (ConstructableRequirementInterface method), 142
- add_requirement() (IntRequirement method), 101
- add_requirement() (LayerListRequirement method), 103
- add_requirement() (ListRequirement method), 105
- add_requirement() (ModuleRequirement method), 106
- add_requirement() (MultiRequirement method), 108
- add_requirement() (PluginRequirement method), 109
- add_requirement() (RequirementInterface method), 145
- add_requirement() (SimpleTypeRequirement method), 147
- add_requirement() (StringRequirement method), 111
- add_requirement() (SymbolTableRequirement method), 113
- add_requirement() (TranslationLayerRequirement method), 114
- add_requirement() (URIRequirement method), 116
- add_requirement() (VersionRequirement method), 117
- add_subparsers() (HelpfulArgParser method), 67
- address (SymbolInterface property), 176
- address_mask (AVMLLayer property), 182
- address_mask (BufferDataLayer property), 225
- address_mask (DataLayerInterface property), 154
- address_mask (Elf64Layer property), 191

- `address_mask` (*FileLayer* property), 227
 - `address_mask` (*Intel* property), 195
 - `address_mask` (*Intel32e* property), 197
 - `address_mask` (*IntelPAE* property), 200
 - `address_mask` (*LimeLayer* property), 214
 - `address_mask` (*LinearlyMappedLayer* property), 217
 - `address_mask` (*NonLinearlySegmentedLayer* property), 239
 - `address_mask` (*PdbMSFStream* property), 220
 - `address_mask` (*PdbMultiStreamFormat* property), 222
 - `address_mask` (*QemuSuspendLayer* property), 231
 - `address_mask` (*RegistryHive* property), 234
 - `address_mask` (*SegmentedLayer* property), 241
 - `address_mask` (*TranslationLayerInterface* property), 158
 - `address_mask` (*VmwareLayer* property), 243
 - `address_mask` (*WindowsCrashDump32Layer* property), 185
 - `address_mask` (*WindowsCrashDump64Layer* property), 188
 - `address_mask` (*WindowsIntel* property), 203
 - `address_mask` (*WindowsIntel32e* property), 206
 - `address_mask` (*WindowsIntelPAE* property), 208
 - `address_mask` (*WindowsMixin* property), 211
 - `address_mask` (*XenCoreDumpLayer* property), 246
 - `ADS` (class in *volatility3.plugins.windows.mftscan*), 702
 - `AggregateType` (class in *volatility3.framework.objects*), 249
 - `AggregateType.VolTemplateProxy` (class in *volatility3.framework.objects*), 249
 - `all_filters` (*Kevents* attribute), 620
 - `alpassword` (*Hashdump* attribute), 686
 - `antpassword` (*Hashdump* attribute), 686
 - `anum` (*Hashdump* attribute), 686
 - `append()` (*SymbolSpace* method), 317
 - `append()` (*SymbolSpaceInterface* method), 176
 - `aqwerty` (*Hashdump* attribute), 686
 - `args` (*ElfFormatException* attribute), 194
 - `args` (*InvalidAddressException* attribute), 555
 - `args` (*LayerException* attribute), 556
 - `args` (*LimeFormatException* attribute), 214
 - `args` (*MissingModuleException* attribute), 556
 - `args` (*OfflineException* attribute), 556
 - `args` (*PagedInvalidAddressException* attribute), 556
 - `args` (*PDBFormatException* attribute), 220
 - `args` (*PluginRequirementException* attribute), 557
 - `args` (*PluginVersionException* attribute), 557
 - `args` (*RegistryFormatException* attribute), 233
 - `args` (*RegistryInvalidIndex* attribute), 237
 - `args` (*SnappyException* attribute), 185
 - `args` (*SwappedInvalidAddressException* attribute), 557
 - `args` (*SymbolError* attribute), 557
 - `args` (*SymbolSpaceError* attribute), 558
 - `args` (*UnsatisfiedException* attribute), 558
 - `args` (*VmwareFormatException* attribute), 243
 - `args` (*VolatilityException* attribute), 558
 - `args` (*WindowsCrashDumpFormatException* attribute), 191
 - `Array` (class in *volatility3.framework.objects*), 251
 - `Array.VolTemplateProxy` (class in *volatility3.framework.objects*), 251
 - `array_of_pointers()` (in module *volatility3.framework.objects.utility*), 300
 - `array_to_string()` (in module *volatility3.framework.objects.utility*), 300
 - `as_integer_ratio()` (*Bin* method), 305
 - `as_integer_ratio()` (*BitField* method), 254
 - `as_integer_ratio()` (*Boolean* method), 257
 - `as_integer_ratio()` (*Char* method), 265
 - `as_integer_ratio()` (*Enumeration* method), 271
 - `as_integer_ratio()` (*Float* method), 274
 - `as_integer_ratio()` (*Hex* method), 307
 - `as_integer_ratio()` (*Integer* method), 278
 - `as_integer_ratio()` (*Parallelism* method), 120
 - `as_integer_ratio()` (*Pointer* method), 282
 - `as_integer_ratio()` (*PoolType* method), 722
 - `as_integer_ratio()` (*RegKeyFlags* method), 510
 - `as_integer_ratio()` (*TimeLinerType* method), 762
 - `ascending` (*ColumnSortKey* attribute), 169, 301
 - `asdict()` (*TreeNode* method), 303
 - `astuple()` (*CapabilitiesData* method), 562
 - `AUTOMAGIC_CONFIG_PATH` (in module *volatility3.framework.constants*), 119
 - `AutomagicInterface` (class in *volatility3.framework.interfaces.automagic*), 134
 - `available()` (in module *volatility3.framework.automagic*), 69
 - `AVMLLayer` (class in *volatility3.framework.layers.avml*), 182
 - `AVMLStacker` (class in *volatility3.framework.layers.avml*), 184
- ## B
- `BANG` (in module *volatility3.framework.constants*), 119
 - `banner_config_key` (*LinuxSymbolFinder* attribute), 72
 - `banner_config_key` (*MacSymbolFinder* attribute), 75
 - `banner_config_key` (*SymbolFinder* attribute), 90
 - `Banners` (class in *volatility3.plugins.banners*), 753
 - `banners` (*LinuxSymbolFinder* property), 72
 - `banners` (*MacSymbolFinder* property), 75
 - `banners` (*SymbolFinder* property), 90
 - `base_types` (*TreeGrid* attribute), 170, 302
 - `BaseAbsentValue` (class in *volatility3.framework.interfaces.renderers*), 168
 - `BaseSymbolTableInterface` (class in *volatility3.framework.interfaces.symbols*), 172
 - `Bash` (class in *volatility3.plugins.linux.bash*), 559
 - `Bash` (class in *volatility3.plugins.mac.bash*), 608

- BashIntermedSymbols (class in volatility3.framework.symbols.linux.bash), 397
- BigPools (class in volatility3.plugins.windows.bigpools), 655
- Bin (class in volatility3.framework.renderers.format_hints), 305
- bit_count() (Bin method), 305
- bit_count() (BitField method), 254
- bit_count() (Boolean method), 257
- bit_count() (Char method), 266
- bit_count() (Enumeration method), 271
- bit_count() (Hex method), 307
- bit_count() (Integer method), 279
- bit_count() (Parallelism method), 120
- bit_count() (Pointer method), 282
- bit_count() (PoolType method), 723
- bit_count() (RegKeyFlags method), 511
- bit_count() (TimeLinerType method), 762
- bit_length() (Bin method), 306
- bit_length() (BitField method), 254
- bit_length() (Boolean method), 257
- bit_length() (Char method), 266
- bit_length() (Enumeration method), 271
- bit_length() (Hex method), 307
- bit_length() (Integer method), 279
- bit_length() (Parallelism method), 121
- bit_length() (Pointer method), 282
- bit_length() (PoolType method), 723
- bit_length() (RegKeyFlags method), 511
- bit_length() (TimeLinerType method), 763
- BitField (class in volatility3.framework.objects), 253
- BitField.VolTemplateProxy (class in volatility3.framework.objects), 253
- bits_per_register (Intel attribute), 195
- bits_per_register (Intel32e attribute), 197
- bits_per_register (IntelPAE attribute), 200
- bits_per_register (WindowsIntel attribute), 203
- bits_per_register (WindowsIntel32e attribute), 206
- bits_per_register (WindowsIntelPAE attribute), 208
- bits_per_register (WindowsMixin attribute), 211
- Boolean (class in volatility3.framework.objects), 256
- Boolean.VolTemplateProxy (class in volatility3.framework.objects), 256
- BooleanRequirement (class in volatility3.framework.configuration.requirements), 95
- bootkey_perm_table (Hashdump attribute), 686
- bpf_prog (class in volatility3.framework.symbols.linux.extensions), 326
- bpf_prog.VolTemplateProxy (class in volatility3.framework.symbols.linux.extensions), 326
- branch() (HierarchicalDict method), 143
- bt_sock (class in volatility3.framework.symbols.linux.extensions), 328
- bt_sock.VolTemplateProxy (class in volatility3.framework.symbols.linux.extensions), 328
- BufferDataLayer (class in volatility3.framework.layers.physical), 225
- build_configuration() (ADS method), 702
- build_configuration() (AutomagicInterface method), 135
- build_configuration() (AVMLLayer method), 182
- build_configuration() (Banners method), 754
- build_configuration() (Bash method), 559, 608
- build_configuration() (BashIntermedSymbols method), 398
- build_configuration() (BigPools method), 656
- build_configuration() (BufferDataLayer method), 225
- build_configuration() (Cachedump method), 657
- build_configuration() (Callbacks method), 659
- build_configuration() (Capabilities method), 560
- build_configuration() (Check_afinfo method), 563
- build_configuration() (Check_creds method), 565
- build_configuration() (Check_idt method), 566
- build_configuration() (Check_modules method), 568
- build_configuration() (Check_syscall method), 570, 610
- build_configuration() (Check_sysctl method), 612
- build_configuration() (Check_trap_table method), 613
- build_configuration() (CmdLine method), 662
- build_configuration() (ComplexListRequirement method), 100
- build_configuration() (ConfigurableInterface method), 139
- build_configuration() (ConfigurableModule method), 123
- build_configuration() (ConfigurableRequirementInterface method), 140
- build_configuration() (ConfigWriter method), 755
- build_configuration() (ConstructionMagic method), 70
- build_configuration() (Crashinfo method), 664
- build_configuration() (DataLayerInterface method), 154
- build_configuration() (DeviceTree method), 666
- build_configuration() (DllList method), 667
- build_configuration() (DriverIrp method), 669
- build_configuration() (DriverModule method), 671
- build_configuration() (DriverScan method), 673
- build_configuration() (DumpFiles method), 675
- build_configuration() (Elf64Layer method), 191

`build_configuration()` (*Elfs method*), 571
`build_configuration()` (*Envvars method*), 573, 677
`build_configuration()` (*FileLayer method*), 227
`build_configuration()` (*FileScan method*), 679
`build_configuration()` (*FrameworkInfo method*), 757
`build_configuration()` (*GetServiceSIDs method*), 680
`build_configuration()` (*GetSIDs method*), 682
`build_configuration()` (*Handles method*), 684
`build_configuration()` (*Hashdump method*), 686
`build_configuration()` (*HiveList method*), 647
`build_configuration()` (*HiveScan method*), 650
`build_configuration()` (*Ifconfig method*), 615
`build_configuration()` (*Info method*), 689
`build_configuration()` (*Intel method*), 195
`build_configuration()` (*Intel32e method*), 197
`build_configuration()` (*IntelPAE method*), 200
`build_configuration()` (*IntermediateSymbolTable method*), 526
`build_configuration()` (*IOMem method*), 575
`build_configuration()` (*IsfInfo method*), 758
`build_configuration()` (*ISFormatTable method*), 523
`build_configuration()` (*JobLinks method*), 691
`build_configuration()` (*Kauth_listeners method*), 616
`build_configuration()` (*Kauth_scopes method*), 618
`build_configuration()` (*KernelModule method*), 77
`build_configuration()` (*KernelPDBScanner method*), 78
`build_configuration()` (*Kevents method*), 620
`build_configuration()` (*Keyboard_notifiers method*), 577
`build_configuration()` (*Kmsg method*), 580
`build_configuration()` (*LayerListRequirement method*), 103
`build_configuration()` (*LayerStacker method*), 82
`build_configuration()` (*LayerWriter method*), 760
`build_configuration()` (*LdrModules method*), 693
`build_configuration()` (*LimeLayer method*), 214
`build_configuration()` (*LinearlyMappedLayer method*), 217
`build_configuration()` (*LinuxKernelIntermedSymbols method*), 321
`build_configuration()` (*LinuxSymbolFinder method*), 72
`build_configuration()` (*List_Files method*), 622
`build_configuration()` (*Lsadump method*), 694
`build_configuration()` (*Lsmmod method*), 585, 624
`build_configuration()` (*Lsof method*), 587, 625
`build_configuration()` (*MacKernelIntermedSymbols method*), 401
`build_configuration()` (*MacSymbolFinder method*), 75
`build_configuration()` (*Malfind method*), 588, 627, 696
`build_configuration()` (*Maps method*), 592, 632
`build_configuration()` (*MBRScan method*), 698
`build_configuration()` (*Memmap method*), 700
`build_configuration()` (*MFTScan method*), 703
`build_configuration()` (*ModScan method*), 705
`build_configuration()` (*Module method*), 127
`build_configuration()` (*ModuleInterface method*), 151
`build_configuration()` (*ModuleRequirement method*), 106
`build_configuration()` (*Modules method*), 707
`build_configuration()` (*Mount method*), 628
`build_configuration()` (*MountInfo method*), 590
`build_configuration()` (*MutantScan method*), 710
`build_configuration()` (*NetScan method*), 712
`build_configuration()` (*NetStat method*), 714
`build_configuration()` (*Netstat method*), 630
`build_configuration()` (*NonLinearlySegmented-Layer method*), 239
`build_configuration()` (*PdbMSFStream method*), 220
`build_configuration()` (*PdbMultiStreamFormat method*), 222
`build_configuration()` (*PluginInterface method*), 167
`build_configuration()` (*PoolScanner method*), 720
`build_configuration()` (*PrintKey method*), 652
`build_configuration()` (*Privs method*), 724
`build_configuration()` (*PsAux method*), 595
`build_configuration()` (*Psaux method*), 633
`build_configuration()` (*PsList method*), 596, 635, 726
`build_configuration()` (*PsScan method*), 599, 729
`build_configuration()` (*PsTree method*), 601, 639, 731
`build_configuration()` (*QemuSuspendLayer method*), 231
`build_configuration()` (*RegistryHive method*), 234
`build_configuration()` (*SegmentedLayer method*), 241
`build_configuration()` (*Sessions method*), 733
`build_configuration()` (*SizedModule method*), 131
`build_configuration()` (*Skeleton_Key_Check method*), 734
`build_configuration()` (*Socket_filters method*), 640
`build_configuration()` (*Sockstat method*), 603
`build_configuration()` (*SSDT method*), 736
`build_configuration()` (*Strings method*), 738
`build_configuration()` (*SvcScan method*), 740
`build_configuration()` (*SymbolCacheMagic method*), 88
`build_configuration()` (*SymbolFinder method*), 90

- `build_configuration()` (*SymbolTableInterface method*), 178
 - `build_configuration()` (*SymbolTableRequirement method*), 113
 - `build_configuration()` (*SymlinkScan method*), 742
 - `build_configuration()` (*Timeliner method*), 764
 - `build_configuration()` (*Timers method*), 642
 - `build_configuration()` (*TranslationLayerInterface method*), 158
 - `build_configuration()` (*TranslationLayerRequirement method*), 114
 - `build_configuration()` (*Trustedbsd method*), 644
 - `build_configuration()` (*tty_check method*), 605
 - `build_configuration()` (*UserAssist method*), 654
 - `build_configuration()` (*VadInfo method*), 744
 - `build_configuration()` (*VadWalk method*), 746
 - `build_configuration()` (*VadYaraScan method*), 748
 - `build_configuration()` (*VerInfo method*), 750
 - `build_configuration()` (*Version1Format method*), 530
 - `build_configuration()` (*Version2Format method*), 532
 - `build_configuration()` (*Version3Format method*), 535
 - `build_configuration()` (*Version4Format method*), 538
 - `build_configuration()` (*Version5Format method*), 541
 - `build_configuration()` (*Version6Format method*), 544
 - `build_configuration()` (*Version7Format method*), 547
 - `build_configuration()` (*Version8Format method*), 550
 - `build_configuration()` (*VFSevents method*), 645
 - `build_configuration()` (*VirtMap method*), 752
 - `build_configuration()` (*VmaYaraScan method*), 606
 - `build_configuration()` (*VmwareLayer method*), 244
 - `build_configuration()` (*Volshell method*), 52, 55, 58, 61
 - `build_configuration()` (*WindowsCrash-Dump32Layer method*), 185
 - `build_configuration()` (*WindowsCrash-Dump64Layer method*), 188
 - `build_configuration()` (*WindowsIntel method*), 203
 - `build_configuration()` (*WindowsIntel32e method*), 206
 - `build_configuration()` (*WindowsIntelPAE method*), 208
 - `build_configuration()` (*WindowsKernelInter-medSymbols method*), 430
 - `build_configuration()` (*WindowsMixin method*), 211
 - `build_configuration()` (*WinSwapLayers method*), 93
 - `build_configuration()` (*XenCoreDumpLayer method*), 246
 - `build_configuration()` (*YaraScan method*), 765
 - `build_module_collection()` (*SSDT class method*), 736
 - `builtin_constraints()` (*PoolScanner static method*), 720
 - `byteorder` (*DataFormatInfo attribute*), 269
 - `Bytes` (*class in volatility3.framework.objects*), 259
 - `Bytes.VolTemplateProxy` (*class in volatility3.framework.objects*), 259
 - `BytesRequirement` (*class in volatility3.framework.configuration.requirements*), 97
 - `BytesScanner` (*class in volatility3.framework.layers.scanners*), 180
- ## C
- `CACHE_PATH` (*in module volatility3.framework.constants*), 119
 - `CACHE_SQLITE_SCHEMA_VERSION` (*in module volatility3.framework.constants*), 119
 - `cached_strtab` (*elf_sym property*), 396
 - `Cachedump` (*class in volatility3.plugins.windows.cachedump*), 657
 - `CacheManagerInterface` (*class in volatility3.framework.automagic.symbol_cache*), 85
 - `Callbacks` (*class in volatility3.plugins.windows.callbacks*), 659
 - `canonicalize()` (*Intel method*), 195
 - `canonicalize()` (*Intel32e method*), 198
 - `canonicalize()` (*IntelPAE method*), 200
 - `canonicalize()` (*WindowsIntel method*), 203
 - `canonicalize()` (*WindowsIntel32e method*), 206
 - `canonicalize()` (*WindowsIntelPAE method*), 208
 - `canonicalize()` (*WindowsMixin method*), 211
 - `cap_ambient` (*CapabilitiesData attribute*), 562
 - `cap_bset` (*CapabilitiesData attribute*), 562
 - `cap_effective` (*CapabilitiesData attribute*), 562
 - `cap_inheritable` (*CapabilitiesData attribute*), 563
 - `cap_permitted` (*CapabilitiesData attribute*), 563
 - `Capabilities` (*class in volatility3.plugins.linux.capabilities*), 560
 - `capabilities_to_string()` (*kernel_cap_struct class method*), 340
 - `capabilities_to_string()` (*kernel_cap_t class method*), 343
 - `CapabilitiesData` (*class in volatility3.plugins.linux.capabilities*), 562
 - `capitalize()` (*Bytes method*), 260
 - `capitalize()` (*HexBytes method*), 308
 - `capitalize()` (*MultiTypeData method*), 312
 - `capitalize()` (*String method*), 287

`cascadeCloseFile()` (in module `volatility3.framework.layers.resources`), 238

`casefold()` (*String* method), 287

`cast()` (*AggregateType* method), 250

`cast()` (*Array* method), 252

`cast()` (*BitField* method), 254

`cast()` (*Boolean* method), 257

`cast()` (*bpf_prog* method), 327

`cast()` (*bt_sock* method), 328

`cast()` (*Bytes* method), 260

`cast()` (*Char* method), 266

`cast()` (*ClassType* method), 268

`cast()` (*CM_KEY_BODY* method), 503

`cast()` (*CM_KEY_NODE* method), 505

`cast()` (*CM_KEY_VALUE* method), 507

`cast()` (*CMHIVE* method), 501

`cast()` (*CONTROL_AREA* method), 433

`cast()` (*cred* method), 330

`cast()` (*dentry* method), 332

`cast()` (*DEVICE_OBJECT* method), 435

`cast()` (*DRIVER_OBJECT* method), 437

`cast()` (*elf* method), 392

`cast()` (*elf_phdr* method), 394

`cast()` (*elf_sym* method), 396

`cast()` (*Enumeration* method), 271

`cast()` (*EPROCESS* method), 440

`cast()` (*ETHREAD* method), 442

`cast()` (*EX_FAST_REF* method), 444

`cast()` (*ExecutiveObject* method), 490

`cast()` (*FILE_OBJECT* method), 446

`cast()` (*fileglob* method), 406

`cast()` (*files_struct* method), 334

`cast()` (*Float* method), 274

`cast()` (*fs_struct* method), 336

`cast()` (*Function* method), 277

`cast()` (*GenericIntelProcess* method), 320

`cast()` (*hist_entry* method), 390

`cast()` (*HMAP_ENTRY* method), 509

`cast()` (*ifnet* method), 407

`cast()` (*IMAGE_DOS_HEADER* method), 485

`cast()` (*IMAGE_NT_HEADERS* method), 488

`cast()` (*inet_sock* method), 338

`cast()` (*inpcb* method), 409

`cast()` (*Integer* method), 279

`cast()` (*kauth_scope* method), 411

`cast()` (*KDDEBUGGER_DATA64* method), 473

`cast()` (*kernel_cap_struct* method), 340

`cast()` (*kernel_cap_t* method), 343

`cast()` (*KMUTANT* method), 448

`cast()` (*kobject* method), 345

`cast()` (*KSYSTEM_TIME* method), 450

`cast()` (*KTHREAD* method), 452

`cast()` (*LIST_ENTRY* method), 454

`cast()` (*list_head* method), 347

`cast()` (*maple_tree* method), 349

`cast()` (*MFTAttribute* method), 480

`cast()` (*MFTEntry* method), 482

`cast()` (*MFTFileName* method), 483

`cast()` (*mm_struct* method), 351

`cast()` (*MMVAD* method), 456

`cast()` (*MMVAD_SHORT* method), 458

`cast()` (*mnt_namespace* method), 353

`cast()` (*module* method), 355

`cast()` (*mount* method), 358

`cast()` (*net* method), 361

`cast()` (*netlink_sock* method), 363

`cast()` (*OBJECT_HEADER* method), 492

`cast()` (*OBJECT_SYMBOLIC_LINK* method), 461

`cast()` (*ObjectInterface* method), 162

`cast()` (*packet_sock* method), 365

`cast()` (*PARTITION_ENTRY* method), 475

`cast()` (*PARTITION_TABLE* method), 478

`cast()` (*Pointer* method), 282

`cast()` (*POOL_HEADER* method), 494

`cast()` (*POOL_HEADER_VISTA* method), 496

`cast()` (*POOL_TRACKER_BIG_PAGES* method), 499

`cast()` (*PrimitiveObject* method), 285

`cast()` (*proc* method), 413

`cast()` (*qstr* method), 367

`cast()` (*queue_entry* method), 415

`cast()` (*SERVICE_HEADER* method), 513

`cast()` (*SERVICE_RECORD* method), 515

`cast()` (*SHARED_CACHE_MAP* method), 463

`cast()` (*sock* method), 369

`cast()` (*sockaddr* method), 417

`cast()` (*sockaddr_dl* method), 419

`cast()` (*socket* method), 370, 420

`cast()` (*String* method), 287

`cast()` (*struct_file* method), 372

`cast()` (*StructType* method), 293

`cast()` (*SUMMARY_DUMP* method), 471

`cast()` (*super_block* method), 375

`cast()` (*sysctl_oid* method), 422

`cast()` (*task_struct* method), 377

`cast()` (*TOKEN* method), 465

`cast()` (*UNICODE_STRING* method), 467

`cast()` (*UnionType* method), 295

`cast()` (*unix_sock* method), 379

`cast()` (*VACB* method), 469

`cast()` (*vfsmount* method), 381

`cast()` (*vm_area_struct* method), 384

`cast()` (*vm_map_entry* method), 424

`cast()` (*vm_map_object* method), 426

`cast()` (*vnode* method), 428

`cast()` (*Void* method), 297

`cast()` (*vsock_sock* method), 387

`cast()` (*xdp_sock* method), 388

`center()` (*Bytes* method), 260

- center() (*HexBytes* method), 308
- center() (*MultiTypeData* method), 312
- center() (*String* method), 287
- change_kernel() (*Volshell* method), 53, 55, 58, 61
- change_layer() (*Volshell* method), 53, 55, 58, 61
- change_process() (*Volshell* method), 61
- change_symbol_table() (*Volshell* method), 53, 55, 58, 61
- change_task() (*Volshell* method), 55, 58
- CHANGED (*TimeLinerType* attribute), 762
- Char (class in *volatility3.framework.objects*), 265
- Char.VolTemplateProxy (class in *volatility3.framework.objects*), 265
- Check_afinfo (class in *volatility3.plugins.linux.check_afinfo*), 563
- Check_creds (class in *volatility3.plugins.linux.check_creds*), 565
- check_cycles() (*LayerContainer* method), 157
- check_header() (*WindowsCrashDump32Layer* class method), 185
- check_header() (*WindowsCrashDump64Layer* class method), 188
- Check_idt (class in *volatility3.plugins.linux.check_idt*), 566
- check_kernel_offset() (*KernelPDBScanner* method), 79
- Check_modules (class in *volatility3.plugins.linux.check_modules*), 568
- Check_syscall (class in *volatility3.plugins.linux.check_syscall*), 569
- Check_syscall (class in *volatility3.plugins.mac.check_syscall*), 610
- Check_sysctl (class in *volatility3.plugins.mac.check_sysctl*), 612
- Check_trap_table (class in *volatility3.plugins.mac.check_trap_table*), 613
- child_template() (*AggregateType.VolTemplateProxy* class method), 249
- child_template() (*Array.VolTemplateProxy* class method), 251
- child_template() (*BitField.VolTemplateProxy* class method), 253
- child_template() (*Boolean.VolTemplateProxy* class method), 256
- child_template() (*bpf_prog.VolTemplateProxy* class method), 326
- child_template() (*bt_sock.VolTemplateProxy* class method), 328
- child_template() (*Bytes.VolTemplateProxy* class method), 259
- child_template() (*Char.VolTemplateProxy* class method), 265
- child_template() (*ClassType.VolTemplateProxy* class method), 268
- child_template() (*CM_KEY_BODY.VolTemplateProxy* class method), 502
- child_template() (*CM_KEY_NODE.VolTemplateProxy* class method), 504
- child_template() (*CM_KEY_VALUE.VolTemplateProxy* class method), 507
- child_template() (*CMHIVE.VolTemplateProxy* class method), 500
- child_template() (*CONTROL_AREA.VolTemplateProxy* class method), 433
- child_template() (*cred.VolTemplateProxy* class method), 330
- child_template() (*dentry.VolTemplateProxy* class method), 332
- child_template() (*DEVICE_OBJECT.VolTemplateProxy* class method), 435
- child_template() (*DRIVER_OBJECT.VolTemplateProxy* class method), 437
- child_template() (*elf.VolTemplateProxy* class method), 392
- child_template() (*elf_phdr.VolTemplateProxy* class method), 394
- child_template() (*elf_sym.VolTemplateProxy* class method), 396
- child_template() (*Enumeration.VolTemplateProxy* class method), 270
- child_template() (*EPROCESS.VolTemplateProxy* class method), 439
- child_template() (*ETHREAD.VolTemplateProxy* class method), 442
- child_template() (*EX_FAST_REF.VolTemplateProxy* class method), 444
- child_template() (*ExecutiveObject.VolTemplateProxy* class method), 490
- child_template() (*FILE_OBJECT.VolTemplateProxy* class method), 445
- child_template() (*fileglob.VolTemplateProxy* class method), 405
- child_template() (*files_struct.VolTemplateProxy* class method), 334
- child_template() (*Float.VolTemplateProxy* class method), 274
- child_template() (*fs_struct.VolTemplateProxy* class method), 336
- child_template() (*Function.VolTemplateProxy* class method), 276
- child_template() (*GenericIntelProcess.VolTemplateProxy* class method), 319
- child_template() (*hist_entry.VolTemplateProxy* class method), 390
- child_template() (*HMAP_ENTRY.VolTemplateProxy* class method), 508

<code>child_template()</code> (<i>ifnet.VolTemplateProxy class method</i>), 407	<code>child_template()</code> (<i>module.VolTemplateProxy class method</i>), 355
<code>child_template()</code> (<i>IMAGE_DOS_HEADER.VolTemplateProxy class method</i>), 485	<code>child_template()</code> (<i>mount.VolTemplateProxy class method</i>), 357
<code>child_template()</code> (<i>IMAGE_NT_HEADERS.VolTemplateProxy class method</i>), 488	<code>child_template()</code> (<i>net.VolTemplateProxy class method</i>), 361
<code>child_template()</code> (<i>inet_sock.VolTemplateProxy class method</i>), 337	<code>child_template()</code> (<i>netlink_sock.VolTemplateProxy class method</i>), 363
<code>child_template()</code> (<i>inpcb.VolTemplateProxy class method</i>), 409	<code>child_template()</code> (<i>OBJECT_HEADER.VolTemplateProxy class method</i>), 491
<code>child_template()</code> (<i>Integer.VolTemplateProxy class method</i>), 278	<code>child_template()</code> (<i>OBJECT_SYMBOLIC_LINK.VolTemplateProxy class method</i>), 460
<code>child_template()</code> (<i>kauth_scope.VolTemplateProxy class method</i>), 410	<code>child_template()</code> (<i>ObjectInterface.VolTemplateProxy class method</i>), 162
<code>child_template()</code> (<i>KDDEBUGGER_DATA64.VolTemplateProxy class method</i>), 473	<code>child_template()</code> (<i>ObjectTemplate method</i>), 298
<code>child_template()</code> (<i>kernel_cap_struct.VolTemplateProxy class method</i>), 339	<code>child_template()</code> (<i>packet_sock.VolTemplateProxy class method</i>), 364
<code>child_template()</code> (<i>kernel_cap_t.VolTemplateProxy class method</i>), 342	<code>child_template()</code> (<i>PARTITION_ENTRY.VolTemplateProxy class method</i>), 474
<code>child_template()</code> (<i>KMUTANT.VolTemplateProxy class method</i>), 447	<code>child_template()</code> (<i>PARTITION_TABLE.VolTemplateProxy class method</i>), 477
<code>child_template()</code> (<i>kobject.VolTemplateProxy class method</i>), 345	<code>child_template()</code> (<i>Pointer.VolTemplateProxy class method</i>), 281
<code>child_template()</code> (<i>KSYSTEM_TIME.VolTemplateProxy class method</i>), 449	<code>child_template()</code> (<i>POOL_HEADER.VolTemplateProxy class method</i>), 494
<code>child_template()</code> (<i>KTHREAD.VolTemplateProxy class method</i>), 451	<code>child_template()</code> (<i>POOL_HEADER_VISTA.VolTemplateProxy class method</i>), 496
<code>child_template()</code> (<i>LIST_ENTRY.VolTemplateProxy class method</i>), 453	<code>child_template()</code> (<i>POOL_TRACKER_BIG_PAGES.VolTemplateProxy class method</i>), 498
<code>child_template()</code> (<i>list_head.VolTemplateProxy class method</i>), 347	<code>child_template()</code> (<i>PrimitiveObject.VolTemplateProxy class method</i>), 284
<code>child_template()</code> (<i>maple_tree.VolTemplateProxy class method</i>), 349	<code>child_template()</code> (<i>proc.VolTemplateProxy class method</i>), 412
<code>child_template()</code> (<i>MFTAttribute.VolTemplateProxy class method</i>), 479	<code>child_template()</code> (<i>qstr.VolTemplateProxy class method</i>), 366
<code>child_template()</code> (<i>MFTEntry.VolTemplateProxy class method</i>), 481	<code>child_template()</code> (<i>queue_entry.VolTemplateProxy class method</i>), 414
<code>child_template()</code> (<i>MFTFileName.VolTemplateProxy class method</i>), 483	<code>child_template()</code> (<i>ReferenceTemplate method</i>), 299
<code>child_template()</code> (<i>mm_struct.VolTemplateProxy class method</i>), 351	<code>child_template()</code> (<i>SERVICE_VICE_HEADER.VolTemplateProxy class method</i>), 513
<code>child_template()</code> (<i>MMVAD.VolTemplateProxy class method</i>), 455	<code>child_template()</code> (<i>SERVICE_VICE_RECORD.VolTemplateProxy class method</i>), 514
<code>child_template()</code> (<i>MMVAD_SHORT.VolTemplateProxy class method</i>), 458	<code>child_template()</code> (<i>SHARED_CACHE_MAP.VolTemplateProxy class method</i>), 462
<code>child_template()</code> (<i>mnt_namespace.VolTemplateProxy class method</i>), 353	<code>child_template()</code> (<i>sock.VolTemplateProxy class method</i>), 368
	<code>child_template()</code> (<i>sockaddr.VolTemplateProxy class method</i>), 416

`child_template()` (*sockaddr_dl.VolTemplateProxy class method*), 418
`child_template()` (*socket.VolTemplateProxy class method*), 370, 420
`child_template()` (*String.VolTemplateProxy class method*), 286
`child_template()` (*struct_file.VolTemplateProxy class method*), 372
`child_template()` (*StructType.VolTemplateProxy class method*), 293
`child_template()` (*SUMMARY_DUMP.VolTemplateProxy class method*), 471
`child_template()` (*super_block.VolTemplateProxy class method*), 374
`child_template()` (*SymbolSpace.UnresolvedTemplate method*), 316
`child_template()` (*sysctl_oid.VolTemplateProxy class method*), 422
`child_template()` (*task_struct.VolTemplateProxy class method*), 376
`child_template()` (*Template method*), 164
`child_template()` (*TOKEN.VolTemplateProxy class method*), 465
`child_template()` (*UNICODE_STRING.VolTemplateProxy class method*), 467
`child_template()` (*UnionType.VolTemplateProxy class method*), 294
`child_template()` (*unix_sock.VolTemplateProxy class method*), 379
`child_template()` (*VACB.VolTemplateProxy class method*), 469
`child_template()` (*vfsmount.VolTemplateProxy class method*), 380
`child_template()` (*vm_area_struct.VolTemplateProxy class method*), 384
`child_template()` (*vm_map_entry.VolTemplateProxy class method*), 424
`child_template()` (*vm_map_object.VolTemplateProxy class method*), 426
`child_template()` (*vnnode.VolTemplateProxy class method*), 428
`child_template()` (*Void.VolTemplateProxy class method*), 296
`child_template()` (*vssock_sock.VolTemplateProxy class method*), 386
`child_template()` (*xdp_sock.VolTemplateProxy class method*), 388
`children` (*ObjectTemplate property*), 298
`children` (*ReferenceTemplate property*), 299
`children` (*SymbolSpace.UnresolvedTemplate property*), 317
`children` (*Template property*), 164
`children()` (*AggregateType.VolTemplateProxy class method*), 249
`children()` (*Array.VolTemplateProxy class method*), 251
`children()` (*BitField.VolTemplateProxy class method*), 253
`children()` (*Boolean.VolTemplateProxy class method*), 256
`children()` (*bpf_prog.VolTemplateProxy class method*), 326
`children()` (*bt_sock.VolTemplateProxy class method*), 328
`children()` (*Bytes.VolTemplateProxy class method*), 259
`children()` (*Char.VolTemplateProxy class method*), 265
`children()` (*ClassType.VolTemplateProxy class method*), 268
`children()` (*CM_KEY_BODY.VolTemplateProxy class method*), 503
`children()` (*CM_KEY_NODE.VolTemplateProxy class method*), 504
`children()` (*CM_KEY_VALUE.VolTemplateProxy class method*), 507
`children()` (*CMHIVE.VolTemplateProxy class method*), 500
`children()` (*CONTROL_AREA.VolTemplateProxy class method*), 433
`children()` (*cred.VolTemplateProxy class method*), 330
`children()` (*dentry.VolTemplateProxy class method*), 332
`children()` (*DEVICE_OBJECT.VolTemplateProxy class method*), 435
`children()` (*DRIVER_OBJECT.VolTemplateProxy class method*), 437
`children()` (*elf.VolTemplateProxy class method*), 392
`children()` (*elf_phdr.VolTemplateProxy class method*), 394
`children()` (*elf_sym.VolTemplateProxy class method*), 396
`children()` (*Enumeration.VolTemplateProxy class method*), 270
`children()` (*EPROCESS.VolTemplateProxy class method*), 439
`children()` (*ETHREAD.VolTemplateProxy class method*), 442
`children()` (*EX_FAST_REF.VolTemplateProxy class method*), 444
`children()` (*ExecutiveObject.VolTemplateProxy class method*), 490
`children()` (*FILE_OBJECT.VolTemplateProxy class method*), 446
`children()` (*fileglob.VolTemplateProxy class method*), 405
`children()` (*files_struct.VolTemplateProxy class*

- method*), 334
- `children()` (*Float.VolTemplateProxy class method*), 274
- `children()` (*fs_struct.VolTemplateProxy class method*), 336
- `children()` (*Function.VolTemplateProxy class method*), 276
- `children()` (*GenericIntelProcess.VolTemplateProxy class method*), 319
- `children()` (*hist_entry.VolTemplateProxy class method*), 390
- `children()` (*HMAP_ENTRY.VolTemplateProxy class method*), 509
- `children()` (*ifnet.VolTemplateProxy class method*), 407
- `children()` (*IMAGE_DOS_HEADER.VolTemplateProxy class method*), 485
- `children()` (*IMAGE_NT_HEADERS.VolTemplateProxy class method*), 488
- `children()` (*inet_sock.VolTemplateProxy class method*), 337
- `children()` (*inpcb.VolTemplateProxy class method*), 409
- `children()` (*Integer.VolTemplateProxy class method*), 278
- `children()` (*kauth_scope.VolTemplateProxy class method*), 410
- `children()` (*KDDEBUGGER_DATA64.VolTemplateProxy class method*), 473
- `children()` (*kernel_cap_struct.VolTemplateProxy class method*), 339
- `children()` (*kernel_cap_t.VolTemplateProxy class method*), 342
- `children()` (*KMUTANT.VolTemplateProxy class method*), 448
- `children()` (*kobject.VolTemplateProxy class method*), 345
- `children()` (*KSYSTEM_TIME.VolTemplateProxy class method*), 450
- `children()` (*KTHREAD.VolTemplateProxy class method*), 451
- `children()` (*LIST_ENTRY.VolTemplateProxy class method*), 453
- `children()` (*list_head.VolTemplateProxy class method*), 347
- `children()` (*maple_tree.VolTemplateProxy class method*), 349
- `children()` (*MFTAttribute.VolTemplateProxy class method*), 479
- `children()` (*MFTEntry.VolTemplateProxy class method*), 481
- `children()` (*MFTFileName.VolTemplateProxy class method*), 483
- `children()` (*mm_struct.VolTemplateProxy class method*), 351
- `children()` (*MMVAD.VolTemplateProxy class method*), 455
- `children()` (*MMVAD_SHORT.VolTemplateProxy class method*), 458
- `children()` (*mnt_namespace.VolTemplateProxy class method*), 353
- `children()` (*module.VolTemplateProxy class method*), 355
- `children()` (*mount.VolTemplateProxy class method*), 357
- `children()` (*net.VolTemplateProxy class method*), 361
- `children()` (*netlink_sock.VolTemplateProxy class method*), 363
- `children()` (*OBJECT_HEADER.VolTemplateProxy class method*), 492
- `children()` (*OBJECT_SYMBOLIC_LINK.VolTemplateProxy class method*), 460
- `children()` (*ObjectInterface.VolTemplateProxy class method*), 162
- `children()` (*packet_sock.VolTemplateProxy class method*), 364
- `children()` (*PARTITION_ENTRY.VolTemplateProxy class method*), 474
- `children()` (*PARTITION_TABLE.VolTemplateProxy class method*), 477
- `children()` (*Pointer.VolTemplateProxy class method*), 281
- `children()` (*POOL_HEADER.VolTemplateProxy class method*), 494
- `children()` (*POOL_HEADER_VISTA.VolTemplateProxy class method*), 496
- `children()` (*POOL_TRACKER_BIG_PAGES.VolTemplateProxy class method*), 498
- `children()` (*PrimitiveObject.VolTemplateProxy class method*), 284
- `children()` (*proc.VolTemplateProxy class method*), 412
- `children()` (*qstr.VolTemplateProxy class method*), 366
- `children()` (*queue_entry.VolTemplateProxy class method*), 414
- `children()` (*SERVICE_HEADER.VolTemplateProxy class method*), 513
- `children()` (*SERVICE_RECORD.VolTemplateProxy class method*), 515
- `children()` (*SHARED_CACHE_MAP.VolTemplateProxy class method*), 463
- `children()` (*sock.VolTemplateProxy class method*), 368
- `children()` (*sockaddr.VolTemplateProxy class method*), 416
- `children()` (*sockaddr_dl.VolTemplateProxy class method*), 418
- `children()` (*socket.VolTemplateProxy class method*), 370, 420
- `children()` (*String.VolTemplateProxy class method*),

- 286
- children() (*struct_file.VolTemplateProxy class method*), 372
- children() (*StructType.VolTemplateProxy class method*), 293
- children() (*SUMMARY_DUMP.VolTemplateProxy class method*), 471
- children() (*super_block.VolTemplateProxy class method*), 374
- children() (*sysctl_oid.VolTemplateProxy class method*), 422
- children() (*task_struct.VolTemplateProxy class method*), 376
- children() (*TOKEN.VolTemplateProxy class method*), 465
- children() (*TreeGrid method*), 170, 302
- children() (*UNICODE_STRING.VolTemplateProxy class method*), 467
- children() (*UnionType.VolTemplateProxy class method*), 294
- children() (*unix_sock.VolTemplateProxy class method*), 379
- children() (*VACB.VolTemplateProxy class method*), 469
- children() (*vfsmount.VolTemplateProxy class method*), 381
- children() (*vm_area_struct.VolTemplateProxy class method*), 384
- children() (*vm_map_entry.VolTemplateProxy class method*), 424
- children() (*vm_map_object.VolTemplateProxy class method*), 426
- children() (*vnnode.VolTemplateProxy class method*), 428
- children() (*Void.VolTemplateProxy class method*), 296
- children() (*vsock_sock.VolTemplateProxy class method*), 386
- children() (*xdp_sock.VolTemplateProxy class method*), 388
- ChoiceRequirement (*class in volatility3.framework.configuration.requirements*), 98
- choices (*Enumeration property*), 271
- choices (*Flags property*), 555
- choose_automagic() (*in module volatility3.framework.automagic*), 69
- choose_os_stackercs() (*in module volatility3.framework.automagic.stackercs*), 85
- class_subclasses() (*in module volatility3.framework*), 68
- classproperty (*class in volatility3*), 47
- ClassRequirement (*class in volatility3.framework.interfaces.configuration*), 137
- ClassType (*class in volatility3.framework.objects*), 268
- ClassType.VolTemplateProxy (*class in volatility3.framework.objects*), 268
- clear_cache() (*in module volatility3.framework*), 68
- clear_symbol_cache() (*BaseSymbolTableInterface method*), 172
- clear_symbol_cache() (*BashIntermedSymbols method*), 398
- clear_symbol_cache() (*IntermediateSymbolTable method*), 526
- clear_symbol_cache() (*ISFormatTable method*), 523
- clear_symbol_cache() (*LinuxKernelIntermedSymbols method*), 321
- clear_symbol_cache() (*MacKernelIntermedSymbols method*), 401
- clear_symbol_cache() (*NativeTable method*), 553
- clear_symbol_cache() (*NativeTableInterface method*), 174
- clear_symbol_cache() (*SymbolSpace method*), 317
- clear_symbol_cache() (*SymbolSpaceInterface method*), 176
- clear_symbol_cache() (*SymbolTableInterface method*), 178
- clear_symbol_cache() (*Version1Format method*), 530
- clear_symbol_cache() (*Version2Format method*), 533
- clear_symbol_cache() (*Version3Format method*), 535
- clear_symbol_cache() (*Version4Format method*), 538
- clear_symbol_cache() (*Version5Format method*), 541
- clear_symbol_cache() (*Version6Format method*), 544
- clear_symbol_cache() (*Version7Format method*), 547
- clear_symbol_cache() (*Version8Format method*), 550
- clear_symbol_cache() (*WindowsKernelIntermedSymbols method*), 430
- CLI_NAME (*CommandLine attribute*), 48
- CLI_NAME (*VolShell attribute*), 49
- CLIRenderer (*class in volatility3.cli.text_renderer*), 64
- clone() (*Context method*), 126
- clone() (*ContextInterface method*), 149
- clone() (*HierarchicalDict method*), 144
- clone() (*ObjectTemplate method*), 298
- clone() (*ReferenceTemplate method*), 299
- clone() (*SymbolSpace.UnresolvedTemplate method*), 317
- clone() (*Template method*), 164
- close() (*FileHandlerInterface method*), 165
- close() (*JarHandler method*), 237
- close() (*NullFileHandler method*), 50
- close() (*OfflineHandler method*), 237
- close() (*VolatilityHandler method*), 238
- closed (*FileHandlerInterface attribute*), 165
- closed (*NullFileHandler attribute*), 50
- cls (*ClassRequirement property*), 137
- CM_KEY_BODY (*class in volatility3.framework.symbols.windows.extensions.registry*),

502
CM_KEY_BODY.VolTemplateProxy (class in volatility3.framework.symbols.windows.extensions.registered_classes), 502
CM_KEY_NODE (class in volatility3.framework.symbols.windows.extensions.registered_classes), 504
CM_KEY_NODE.VolTemplateProxy (class in volatility3.framework.symbols.windows.extensions.registered_classes), 504
CM_KEY_VALUE (class in volatility3.framework.symbols.windows.extensions.registered_classes), 506
CM_KEY_VALUE.VolTemplateProxy (class in volatility3.framework.symbols.windows.extensions.registered_classes), 506
CmdLine (class in volatility3.plugins.windows.cmdline), 662
CMHIVE (class in volatility3.framework.symbols.windows.extensions.registered_classes), 500
CMHIVE.VolTemplateProxy (class in volatility3.framework.symbols.windows.extensions.registered_classes), 500
Column (class in volatility3.framework.interfaces.renderers), 168
columns (TreeGrid property), 170, 302
ColumnSortKey (class in volatility3.framework.interfaces.renderers), 169
ColumnSortKey (class in volatility3.framework.renderers), 301
comm (TaskData attribute), 563
CommandLine (class in volatility3.cli), 48
ComplexListRequirement (class in volatility3.framework.configuration.requirements), 99
config (ADS property), 702
config (AutomagicInterface property), 135
config (AVMLLayer property), 182
config (Banners property), 754
config (Bash property), 559, 609
config (BashIntermedSymbols property), 398
config (BigPools property), 656
config (BufferDataLayer property), 225
config (Cachedump property), 657
config (Callbacks property), 659
config (Capabilities property), 561
config (Check_afinfo property), 563
config (Check_creds property), 565
config (Check_idt property), 566
config (Check_modules property), 568
config (Check_syscall property), 570, 610
config (Check_sysctl property), 612
config (Check_trap_table property), 613
config (CmdLine property), 662
config (ConfigurableInterface property), 139
config (ConfigurableModule property), 123
config (ConfigWriter property), 755
config (ConstructionMagic property), 70
config (Context property), 126
config (ContextInterface property), 149
config (Crashinfo property), 664
config (DataLayerInterface property), 154
config (DeviceTree property), 666
config (DllList property), 668
config (DriverIrp property), 670
config (DriverModule property), 671
config (DriverScan property), 673
config (DumpFiles property), 675
config (Elf64Layer property), 192
config (Elfs property), 571
config (Envvars property), 573, 677
config (FileLayer property), 228
config (FileScan property), 679
config (FrameworkInfo property), 757
config (GetServiceSIDs property), 681
config (GetSIDs property), 682
config (Handles property), 684
config (Hashdump property), 686
config (HiveList property), 648
config (HiveScan property), 650
config (Ifconfig property), 615
config (Info property), 689
config (Intel property), 195
config (Intel32e property), 198
config (IntelPAE property), 200
config (IntermediateSymbolTable property), 527
config (IOMem property), 575
config (IsfInfo property), 758
config (ISFormatTable property), 523
config (JobLinks property), 691
config (Kauth_listeners property), 617
config (Kauth_scopes property), 618
config (KernelModule property), 77
config (KernelPDBScanner property), 79
config (Kevents property), 620
config (Keyboard_notifiers property), 577
config (Kmsg property), 580
config (LayerStacker property), 82
config (LayerWriter property), 760
config (LdrModules property), 693
config (LimeLayer property), 214
config (LinearlyMappedLayer property), 217
config (LinuxKernelIntermedSymbols property), 321
config (LinuxSymbolFinder property), 73
config (List_Files property), 622
config (Lsadump property), 695
config (Lsmmod property), 585, 624

- config (*Lsof* property), 587, 625
- config (*MacKernellIntermedSymbols* property), 401
- config (*MacSymbolFinder* property), 75
- config (*Malfind* property), 589, 627, 696
- config (*Maps* property), 593, 632
- config (*MBRScan* property), 699
- config (*Memmap* property), 700
- config (*MFTScan* property), 703
- config (*ModScan* property), 705
- config (*Module* property), 127
- config (*ModuleInterface* property), 151
- config (*Modules* property), 708
- config (*Mount* property), 629
- config (*MountInfo* property), 590
- config (*MutantScan* property), 710
- config (*NetScan* property), 712
- config (*NetStat* property), 715
- config (*Netstat* property), 630
- config (*NonLinearlySegmentedLayer* property), 239
- config (*PdbMSFStream* property), 220
- config (*PdbMultiStreamFormat* property), 223
- config (*PluginInterface* property), 167
- config (*PoolScanner* property), 720
- config (*PrintKey* property), 652
- config (*Privs* property), 724
- config (*PsAux* property), 595
- config (*Psaux* property), 634
- config (*PsList* property), 596, 635, 726
- config (*PsScan* property), 599, 729
- config (*PsTree* property), 601, 639, 731
- config (*QemuSuspendLayer* property), 231
- config (*RegistryHive* property), 234
- config (*SegmentedLayer* property), 241
- config (*Sessions* property), 733
- config (*SizedModule* property), 131
- config (*Skeleton_Key_Check* property), 735
- config (*Socket_filters* property), 640
- config (*Sockstat* property), 603
- config (*SSDT* property), 736
- config (*Strings* property), 738
- config (*SvcScan* property), 740
- config (*SymbolCacheMagic* property), 89
- config (*SymbolFinder* property), 91
- config (*SymbolTableInterface* property), 178
- config (*SymlinkScan* property), 742
- config (*Timeliner* property), 764
- config (*Timers* property), 642
- config (*TranslationLayerInterface* property), 158
- config (*Trustedbsd* property), 644
- config (*tty_check* property), 605
- config (*UserAssist* property), 654
- config (*VadInfo* property), 744
- config (*VadWalk* property), 746
- config (*VadYaraScan* property), 748
- config (*VerInfo* property), 750
- config (*Version1Format* property), 530
- config (*Version2Format* property), 533
- config (*Version3Format* property), 535
- config (*Version4Format* property), 538
- config (*Version5Format* property), 541
- config (*Version6Format* property), 544
- config (*Version7Format* property), 547
- config (*Version8Format* property), 550
- config (*VFSevents* property), 645
- config (*VirtMap* property), 752
- config (*VmaYaraScan* property), 606
- config (*VmwareLayer* property), 244
- config (*Volshell* property), 53, 56, 58, 61
- config (*WindowsCrashDump32Layer* property), 185
- config (*WindowsCrashDump64Layer* property), 188
- config (*WindowsIntel* property), 203
- config (*WindowsIntel32e* property), 206
- config (*WindowsIntelPAE* property), 209
- config (*WindowsKernelIntermedSymbols* property), 430
- config (*WindowsMixin* property), 211
- config (*WinSwapLayers* property), 93
- config (*XenCoreDumpLayer* property), 246
- config (*YaraScan* property), 765
- config_path (*ADS* property), 702
- config_path (*AutomagicInterface* property), 135
- config_path (*AVMLLayer* property), 182
- config_path (*Banners* property), 754
- config_path (*Bash* property), 559, 609
- config_path (*BashIntermedSymbols* property), 398
- config_path (*BigPools* property), 656
- config_path (*BufferDataLayer* property), 225
- config_path (*Cachedump* property), 658
- config_path (*Callbacks* property), 659
- config_path (*Capabilities* property), 561
- config_path (*Check_afinfo* property), 563
- config_path (*Check_creds* property), 565
- config_path (*Check_idt* property), 567
- config_path (*Check_modules* property), 568
- config_path (*Check_syscall* property), 570, 610
- config_path (*Check_sysctl* property), 612
- config_path (*Check_trap_table* property), 614
- config_path (*CmdLine* property), 663
- config_path (*ConfigurableInterface* property), 139
- config_path (*ConfigurableModule* property), 123
- config_path (*ConfigWriter* property), 755
- config_path (*ConstructionMagic* property), 71
- config_path (*Crashinfo* property), 664
- config_path (*DataLayerInterface* property), 154
- config_path (*DeviceTree* property), 666
- config_path (*DllList* property), 668
- config_path (*DriverIrp* property), 670
- config_path (*DriverModule* property), 671
- config_path (*DriverScan* property), 673

`config_path` (*DumpFiles* property), 675
`config_path` (*Elf64Layer* property), 192
`config_path` (*Elfs* property), 571
`config_path` (*Envvars* property), 573, 677
`config_path` (*FileLayer* property), 228
`config_path` (*FileScan* property), 679
`config_path` (*FrameworkInfo* property), 757
`config_path` (*GetServiceSIDs* property), 681
`config_path` (*GetSIDs* property), 682
`config_path` (*Handles* property), 684
`config_path` (*Hashdump* property), 686
`config_path` (*HiveList* property), 648
`config_path` (*HiveScan* property), 650
`config_path` (*Ifconfig* property), 615
`config_path` (*Info* property), 689
`config_path` (*Intel* property), 195
`config_path` (*Intel32e* property), 198
`config_path` (*IntelPAE* property), 200
`config_path` (*IntermediateSymbolTable* property), 527
`config_path` (*IOMem* property), 575
`config_path` (*IsfInfo* property), 759
`config_path` (*ISFormatTable* property), 523
`config_path` (*JobLinks* property), 691
`config_path` (*Kauth_listeners* property), 617
`config_path` (*Kauth_scopes* property), 618
`config_path` (*KernelModule* property), 77
`config_path` (*KernelPDBScanner* property), 79
`config_path` (*Kevents* property), 620
`config_path` (*Keyboard_notifiers* property), 577
`config_path` (*Kmsg* property), 580
`config_path` (*LayerStacker* property), 82
`config_path` (*LayerWriter* property), 760
`config_path` (*LdrModules* property), 693
`config_path` (*LimeLayer* property), 214
`config_path` (*LinearlyMappedLayer* property), 217
`config_path` (*LinuxKernelIntermedSymbols* property), 322
`config_path` (*LinuxSymbolFinder* property), 73
`config_path` (*List_Files* property), 622
`config_path` (*Lsadbump* property), 695
`config_path` (*Lsmode* property), 585, 624
`config_path` (*Lsof* property), 587, 626
`config_path` (*MacKernelIntermedSymbols* property), 401
`config_path` (*MacSymbolFinder* property), 75
`config_path` (*Malfind* property), 589, 627, 697
`config_path` (*Maps* property), 593, 632
`config_path` (*MBRScan* property), 699
`config_path` (*Memmap* property), 700
`config_path` (*MFTScan* property), 703
`config_path` (*ModScan* property), 705
`config_path` (*Module* property), 127
`config_path` (*ModuleInterface* property), 151
`config_path` (*Modules* property), 708
`config_path` (*Mount* property), 629
`config_path` (*MountInfo* property), 590
`config_path` (*MutantScan* property), 710
`config_path` (*NetScan* property), 712
`config_path` (*NetStat* property), 715
`config_path` (*Netstat* property), 630
`config_path` (*NonLinearlySegmentedLayer* property), 239
`config_path` (*PdbMSFStream* property), 220
`config_path` (*PdbMultiStreamFormat* property), 223
`config_path` (*PluginInterface* property), 167
`config_path` (*PoolScanner* property), 720
`config_path` (*PrintKey* property), 652
`config_path` (*Privs* property), 724
`config_path` (*PsAux* property), 595
`config_path` (*Psaux* property), 634
`config_path` (*PsList* property), 597, 635, 726
`config_path` (*PsScan* property), 599, 729
`config_path` (*PsTree* property), 601, 639, 731
`config_path` (*QemuSuspendLayer* property), 231
`config_path` (*RegistryHive* property), 234
`config_path` (*SegmentedLayer* property), 241
`config_path` (*Sessions* property), 733
`config_path` (*SizedModule* property), 131
`config_path` (*Skeleton_Key_Check* property), 735
`config_path` (*Socket_filters* property), 641
`config_path` (*Sockstat* property), 603
`config_path` (*SSDT* property), 737
`config_path` (*Strings* property), 738
`config_path` (*SvcScan* property), 740
`config_path` (*SymbolCacheMagic* property), 89
`config_path` (*SymbolFinder* property), 91
`config_path` (*SymbolTableInterface* property), 178
`config_path` (*SymlinkScan* property), 742
`config_path` (*Timeliner* property), 764
`config_path` (*Timers* property), 642
`config_path` (*TranslationLayerInterface* property), 159
`config_path` (*Trustedbsd* property), 644
`config_path` (*tty_check* property), 605
`config_path` (*UserAssist* property), 654
`config_path` (*VadInfo* property), 744
`config_path` (*VadWalk* property), 746
`config_path` (*VadYaraScan* property), 748
`config_path` (*VerInfo* property), 750
`config_path` (*Version1Format* property), 530
`config_path` (*Version2Format* property), 533
`config_path` (*Version3Format* property), 536
`config_path` (*Version4Format* property), 538
`config_path` (*Version5Format* property), 541
`config_path` (*Version6Format* property), 544
`config_path` (*Version7Format* property), 547
`config_path` (*Version8Format* property), 550
`config_path` (*VFSevents* property), 645
`config_path` (*VirtMap* property), 752

- `config_path` (*VmaYaraScan property*), 607
- `config_path` (*VmwareLayer property*), 244
- `config_path` (*Volshell property*), 53, 56, 58, 61
- `config_path` (*WindowsCrashDump32Layer property*), 185
- `config_path` (*WindowsCrashDump64Layer property*), 188
- `config_path` (*WindowsIntel property*), 203
- `config_path` (*WindowsIntel32e property*), 206
- `config_path` (*WindowsIntelPAE property*), 209
- `config_path` (*WindowsKernelIntermedSymbols property*), 430
- `config_path` (*WindowsMixin property*), 211
- `config_path` (*WinSwapLayers property*), 93
- `config_path` (*XenCoreDumpLayer property*), 247
- `config_path` (*YaraScan property*), 765
- `CONFIG_SEPARATOR` (in module *volatility3.framework.interfaces.configuration*), 137
- `config_value()` (*BooleanRequirement method*), 96
- `config_value()` (*BytesRequirement method*), 97
- `config_value()` (*ChoiceRequirement method*), 98
- `config_value()` (*ClassRequirement method*), 137
- `config_value()` (*ComplexListRequirement method*), 100
- `config_value()` (*ConfigurableRequirementInterface method*), 140
- `config_value()` (*ConstructableRequirementInterface method*), 142
- `config_value()` (*IntRequirement method*), 101
- `config_value()` (*LayerListRequirement method*), 103
- `config_value()` (*ListRequirement method*), 105
- `config_value()` (*ModuleRequirement method*), 106
- `config_value()` (*MultiRequirement method*), 108
- `config_value()` (*PluginRequirement method*), 110
- `config_value()` (*RequirementInterface method*), 145
- `config_value()` (*SimpleTypeRequirement method*), 147
- `config_value()` (*StringRequirement method*), 111
- `config_value()` (*SymbolTableRequirement method*), 113
- `config_value()` (*TranslationLayerRequirement method*), 115
- `config_value()` (*URIRequirement method*), 116
- `config_value()` (*VersionRequirement method*), 118
- `ConfigurableInterface` (class in *volatility3.framework.interfaces.configuration*), 138
- `ConfigurableModule` (class in *volatility3.framework.contexts*), 123
- `ConfigurableRequirementInterface` (class in *volatility3.framework.interfaces.configuration*), 140
- `ConfigWriter` (class in *volatility3.plugins.configwriter*), 755
- `conjugate()` (*Bin method*), 306
- `conjugate()` (*BitField method*), 254
- `conjugate()` (*Boolean method*), 257
- `conjugate()` (*Char method*), 266
- `conjugate()` (*Enumeration method*), 271
- `conjugate()` (*Float method*), 274
- `conjugate()` (*Hex method*), 307
- `conjugate()` (*Integer method*), 279
- `conjugate()` (*Parallelism method*), 121
- `conjugate()` (*Pointer method*), 282
- `conjugate()` (*PoolType method*), 723
- `conjugate()` (*RegKeyFlags method*), 511
- `conjugate()` (*TimeLinerType method*), 763
- `constant_data` (*SymbolInterface property*), 176
- `construct()` (*ComplexListRequirement method*), 100
- `construct()` (*ConstructableRequirementInterface method*), 142
- `construct()` (*LayerListRequirement method*), 103
- `construct()` (*ModuleRequirement method*), 107
- `construct()` (*SymbolTableRequirement method*), 113
- `construct()` (*TranslationLayerRequirement method*), 115
- `construct_locals()` (*Volshell method*), 53, 56, 59, 61
- `construct_plugin()` (in module *volatility3.framework.plugins*), 301
- `ConstructableRequirementInterface` (class in *volatility3.framework.interfaces.configuration*), 141
- `ConstructionMagic` (class in *volatility3.framework.automagic.construct_layers*), 70
- `consume_padding()` (*PdbReader method*), 517
- `consume_type()` (*PdbReader method*), 517
- `container_of()` (*LinuxUtilities class method*), 324
- `context` (*ADS property*), 702
- `context` (*AutomagicInterface property*), 135
- `context` (*AVMLLayer property*), 182
- `context` (*Banners property*), 754
- `context` (*Bash property*), 559, 609
- `context` (*BashIntermedSymbols property*), 398
- `context` (*BigPools property*), 656
- `context` (*BufferDataLayer property*), 225
- `context` (*BytesScanner property*), 180
- `context` (*Cachedump property*), 658
- `context` (*Callbacks property*), 659
- `context` (*Capabilities property*), 561
- `context` (*Check_afinfo property*), 563
- `context` (*Check_creds property*), 565
- `context` (*Check_idt property*), 567
- `context` (*Check_modules property*), 568
- `context` (*Check_syscall property*), 570, 610
- `context` (*Check_sysctl property*), 612
- `context` (*Check_trap_table property*), 614
- `Context` (class in *volatility3.framework.contexts*), 125

context (*CmdLine* property), 663
context (*ConfigurableInterface* property), 139
context (*ConfigurableModule* property), 123
context (*ConfigWriter* property), 755
context (*ConstructionMagic* property), 71
context (*Crashinfo* property), 664
context (*DataLayerInterface* property), 154
context (*DeviceTree* property), 666
context (*DllList* property), 668
context (*DriverIrp* property), 670
context (*DriverModule* property), 671
context (*DriverScan* property), 673
context (*DumpFiles* property), 675
context (*Elf64Layer* property), 192
context (*Elfs* property), 571
context (*Envvars* property), 573, 677
context (*FileLayer* property), 228
context (*FileScan* property), 679
context (*FrameworkInfo* property), 757
context (*GetServiceSIDs* property), 681
context (*GetSIDs* property), 682
context (*Handles* property), 684
context (*Hashdump* property), 687
context (*HiveList* property), 648
context (*HiveScan* property), 650
context (*Ifconfig* property), 615
context (*Info* property), 689
context (*Intel* property), 195
context (*Intel32e* property), 198
context (*IntelPAE* property), 201
context (*IntermediateSymbolTable* property), 527
context (*IOMem* property), 575
context (*IsInfo* property), 759
context (*ISFormatTable* property), 523
context (*JobLinks* property), 692
context (*Kauth_listeners* property), 617
context (*Kauth_scopes* property), 618
context (*KernelModule* property), 77
context (*KernelPDBScanner* property), 79
context (*Kevents* property), 620
context (*Keyboard_notifiers* property), 577
context (*Kmsg* property), 580
context (*LayerStacker* property), 82
context (*LayerWriter* property), 760
context (*LdrModules* property), 693
context (*LimeLayer* property), 214
context (*LinearlyMappedLayer* property), 217
context (*LinuxKernelIntermedSymbols* property), 322
context (*LinuxSymbolFinder* property), 73
context (*List_Files* property), 622
context (*Lsadump* property), 695
context (*Lsmodule* property), 585, 624
context (*Lsof* property), 587, 626
context (*MacKernelIntermedSymbols* property), 401
context (*MacSymbolFinder* property), 75
context (*Malfind* property), 589, 627, 697
context (*Maps* property), 593, 632
context (*MBRScan* property), 699
context (*Memmap* property), 700
context (*MFTScan* property), 704
context (*ModScan* property), 705
context (*Module* property), 127
context (*ModuleInterface* property), 151
context (*Modules* property), 708
context (*Mount* property), 629
context (*MountInfo* property), 590
context (*MultiStringScanner* property), 180
context (*MutantScan* property), 710
context (*NetScan* property), 712
context (*NetStat* property), 715
context (*Netstat* property), 630
context (*NonLinearlySegmentedLayer* property), 239
context (*PageMapScanner* property), 93
context (*PdbMSFStream* property), 220
context (*PdbMultiStreamFormat* property), 223
context (*PdbReader* property), 517
context (*PdbSignatureScanner* property), 522
context (*PluginInterface* property), 167
context (*PoolHeaderScanner* property), 719
context (*PoolScanner* property), 720
context (*PrintKey* property), 652
context (*Privs* property), 724
context (*PsAux* property), 595
context (*Psaux* property), 634
context (*PsList* property), 597, 635, 726
context (*PsScan* property), 599, 729
context (*PsTree* property), 601, 639, 731
context (*QemuSuspendLayer* property), 231
context (*RegExScanner* property), 181
context (*RegistryHive* property), 234
context (*ScannerInterface* property), 158
context (*SegmentedLayer* property), 241
context (*Sessions* property), 733
context (*SizedModule* property), 131
context (*Skeleton_Key_Check* property), 735
context (*Socket_filters* property), 641
context (*Sockstat* property), 603
context (*SSDT* property), 737
context (*Strings* property), 738
context (*SvcScan* property), 740
context (*SymbolCacheMagic* property), 89
context (*SymbolFinder* property), 91
context (*SymbolTableInterface* property), 178
context (*SymlinkScan* property), 742
context (*Timeliner* property), 764
context (*Timers* property), 642
context (*TranslationLayerInterface* property), 159
context (*Trustedbsd* property), 644

- `context` (*tty_check* property), 605
- `context` (*UserAssist* property), 654
- `context` (*VadInfo* property), 744
- `context` (*VadWalk* property), 746
- `context` (*VadYaraScan* property), 748
- `context` (*VerInfo* property), 750
- `context` (*Version1Format* property), 530
- `context` (*Version2Format* property), 533
- `context` (*Version3Format* property), 536
- `context` (*Version4Format* property), 539
- `context` (*Version5Format* property), 542
- `context` (*Version6Format* property), 545
- `context` (*Version7Format* property), 548
- `context` (*Version8Format* property), 551
- `context` (*VFSevents* property), 645
- `context` (*VirtMap* property), 752
- `context` (*VmaYaraScan* property), 607
- `context` (*VmwareLayer* property), 244
- `context` (*Volshell* property), 53, 56, 59, 62
- `context` (*WindowsCrashDump32Layer* property), 185
- `context` (*WindowsCrashDump64Layer* property), 188
- `context` (*WindowsIntel* property), 203
- `context` (*WindowsIntel32e* property), 206
- `context` (*WindowsIntelPAE* property), 209
- `context` (*WindowsKernelIntermedSymbols* property), 430
- `context` (*WindowsMixin* property), 211
- `context` (*WinSwapLayers* property), 93
- `context` (*XenCoreDumpLayer* property), 247
- `context` (*YaraScan* property), 766
- `context` (*YaraScanner* property), 767
- `ContextInterface` (class in *volatility3.framework.interfaces.context*), 148
- `CONTROL_AREA` (class in *volatility3.framework.symbols.windows.extensions*), 432
- `CONTROL_AREA.VolTemplateProxy` (class in *volatility3.framework.symbols.windows.extensions*), 433
- `convert_arg_line_to_args()` (*HelpfulArgParser* method), 67
- `convert_bytes_to_guid()` (*PdbReader* method), 517
- `convert_data_to_value()` (in module *volatility3.framework.objects*), 298
- `convert_fields()` (*PdbReader* method), 517
- `convert_ipv4()` (in module *volatility3.framework.renderers.conversion*), 304
- `convert_ipv6()` (in module *volatility3.framework.renderers.conversion*), 304
- `convert_network_four_tuple()` (in module *volatility3.framework.renderers.conversion*), 304
- `convert_port()` (in module *volatility3.framework.renderers.conversion*), 304
- `convert_value_to_data()` (in module *volatility3.framework.objects*), 298
- `count` (*Array* property), 252
- `count()` (*Bytes* method), 260
- `count()` (*Column* method), 168
- `count()` (*DataFormatInfo* method), 270
- `count()` (*HexBytes* method), 308
- `count()` (*MountInfoData* method), 591
- `count()` (*MultiTypeData* method), 312
- `count()` (*String* method), 287
- `count()` (*TreeNode* method), 171, 303
- `crashdump_json` (*WindowsCrashDump32Layer* attribute), 185
- `crashdump_json` (*WindowsCrashDump64Layer* attribute), 188
- `Crashinfo` (class in *volatility3.plugins.windows.crashinfo*), 664
- `create()` (*BashIntermedSymbols* class method), 398
- `create()` (*ConfigurableModule* class method), 123
- `create()` (*IntermediateSymbolTable* class method), 527
- `create()` (*LinuxKernelIntermedSymbols* class method), 322
- `create()` (*MacKernelIntermedSymbols* class method), 401
- `create()` (*Module* class method), 127
- `create()` (*SizedModule* class method), 131
- `create()` (*WindowsKernelIntermedSymbols* class method), 430
- `create_callback_table()` (*Callbacks* static method), 659
- `create_configurable()` (*Volshell* method), 53, 56, 59, 62
- `create_json_hash()` (in module *volatility3.schemas*), 767
- `create_name_filter()` (*PsList* class method), 726
- `create_netscan_constraints()` (*NetScan* static method), 712
- `create_netscan_symbol_table()` (*NetScan* class method), 712
- `create_pid_filter()` (*PsList* class method), 597, 635, 726
- `create_service_table()` (*SvcScan* static method), 740
- `create_stackers_list()` (*LayerStacker* method), 82
- `create_stream_from_pages()` (*PdbMultiStreamFormat* method), 223
- `create_tcpip_symbol_table()` (*NetStat* class method), 715
- `CREATED` (*TimeLinerType* attribute), 762
- `createservicesid()` (in module *volatility3.plugins.windows.getservicesids*), 682
- `cred` (class in *volatility3.framework.symbols.linux.extensions*), 329

`cred.VolTemplateProxy` (class in `volatility3.framework.symbols.linux.extensions`), 330

`CSVRenderer` (class in `volatility3.cli.text_renderer`), 64

`current_kernel_name` (Volshell property), 53, 56, 59, 62

`current_layer` (Volshell property), 53, 56, 59, 62

`current_symbol_table` (Volshell property), 53, 56, 59, 62

D

`d_ancestor()` (dentry method), 332

`data` (HierarchicalDict property), 144

`DataFormatInfo` (class in `volatility3.framework.objects`), 269

`DataLayerInterface` (class in `volatility3.framework.interfaces.layers`), 154

`decanonicalize()` (Intel method), 195

`decanonicalize()` (Intel32e method), 198

`decanonicalize()` (IntelPAE method), 201

`decanonicalize()` (WindowsIntel method), 203

`decanonicalize()` (WindowsIntel32e method), 206

`decanonicalize()` (WindowsIntelPAE method), 209

`decanonicalize()` (WindowsMixin method), 211

`decode()` (Bytes method), 260

`decode()` (HexBytes method), 308

`decode()` (MultiTypeData method), 312

`decode_data()` (CM_KEY_VALUE method), 507

`decrypt_aes()` (Lsadump class method), 695

`decrypt_hash()` (Cachedump static method), 658

`decrypt_secret()` (Lsadump class method), 695

`decrypt_single_hash()` (Hashdump class method), 687

`decrypt_single_saltied_hash()` (Hashdump class method), 687

`deduplicate()` (ModuleCollection method), 130

`default` (BooleanRequirement property), 96

`default` (BytesRequirement property), 97

`default` (ChoiceRequirement property), 98

`default` (ClassRequirement property), 138

`default` (ComplexListRequirement property), 100

`default` (ConfigurableRequirementInterface property), 140

`default` (ConstructableRequirementInterface property), 142

`default` (IntRequirement property), 102

`default` (LayerListRequirement property), 103

`default` (ListRequirement property), 105

`default` (ModuleRequirement property), 107

`default` (MultiRequirement property), 108

`default` (PluginRequirement property), 110

`default` (RequirementInterface property), 145

`default` (SimpleTypeRequirement property), 147

`default` (StringRequirement property), 112

`default` (SymbolTableRequirement property), 113

`default` (TranslationLayerRequirement property), 115

`default` (URIRequirement property), 116

`default` (VersionRequirement property), 118

`default_block_size` (LayerWriter attribute), 760

`default_open()` (JarHandler static method), 237

`default_open()` (OfflineHandler static method), 237

`del_layer()` (LayerContainer method), 157

`del_type_class()` (BaseSymbolTableInterface method), 172

`del_type_class()` (BashIntermedSymbols method), 399

`del_type_class()` (IntermediateSymbolTable method), 527

`del_type_class()` (ISFormatTable method), 523

`del_type_class()` (LinuxKernelIntermedSymbols method), 322

`del_type_class()` (MacKernelIntermedSymbols method), 402

`del_type_class()` (NativeTable method), 554

`del_type_class()` (NativeTableInterface method), 174

`del_type_class()` (SymbolTableInterface method), 178

`del_type_class()` (Version1Format method), 530

`del_type_class()` (Version2Format method), 533

`del_type_class()` (Version3Format method), 536

`del_type_class()` (Version4Format method), 539

`del_type_class()` (Version5Format method), 542

`del_type_class()` (Version6Format method), 545

`del_type_class()` (Version7Format method), 548

`del_type_class()` (Version8Format method), 551

`del_type_class()` (WindowsKernelIntermedSymbols method), 430

`deleter()` (classproperty method), 47

`denominator` (Bin attribute), 306

`denominator` (BitField attribute), 254

`denominator` (Boolean attribute), 257

`denominator` (Char attribute), 266

`denominator` (Enumeration attribute), 272

`denominator` (Hex attribute), 307

`denominator` (Integer attribute), 279

`denominator` (Parallelism attribute), 121

`denominator` (Pointer attribute), 282

`denominator` (PoolType attribute), 723

`denominator` (RegKeyFlags attribute), 511

`denominator` (TimeLinerType attribute), 763

`dentry` (class in `volatility3.framework.symbols.linux.extensions`), 331

`dentry.VolTemplateProxy` (class in `volatility3.framework.symbols.linux.extensions`), 332

`dependencies` (AVMLLayer property), 182

`dependencies` (BufferDataLayer property), 225

- dependencies (*DataLayerInterface* property), 154
- dependencies (*Elf64Layer* property), 192
- dependencies (*FileLayer* property), 228
- dependencies (*Intel* property), 195
- dependencies (*Intel32e* property), 198
- dependencies (*IntelPAE* property), 201
- dependencies (*LimeLayer* property), 214
- dependencies (*LinearlyMappedLayer* property), 217
- dependencies (*NonLinearlySegmentedLayer* property), 239
- dependencies (*PdbMSFStream* property), 220
- dependencies (*PdbMultiStreamFormat* property), 223
- dependencies (*QemuSuspendLayer* property), 231
- dependencies (*RegistryHive* property), 234
- dependencies (*SegmentedLayer* property), 241
- dependencies (*TranslationLayerInterface* property), 159
- dependencies (*VmwareLayer* property), 244
- dependencies (*WindowsCrashDump32Layer* property), 185
- dependencies (*WindowsCrashDump64Layer* property), 188
- dependencies (*WindowsIntel* property), 203
- dependencies (*WindowsIntel32e* property), 206
- dependencies (*WindowsIntelPAE* property), 209
- dependencies (*WindowsMixin* property), 212
- dependencies (*XenCoreDumpLayer* property), 247
- dereference() (*EX_FAST_REF* method), 444
- dereference() (*Pointer* method), 282
- desc_committed (*DescStateEnum* attribute), 579
- desc_finalized (*DescStateEnum* attribute), 579
- desc_miss (*DescStateEnum* attribute), 579
- desc_reserved (*DescStateEnum* attribute), 579
- desc_reusable (*DescStateEnum* attribute), 580
- DescExitStateEnum (class in *volatility3.plugins.linux.psscan*), 599
- description (*BooleanRequirement* property), 96
- description (*BytesRequirement* property), 97
- description (*ChoiceRequirement* property), 98
- description (*ClassRequirement* property), 138
- description (*ComplexListRequirement* property), 100
- description (*ConfigurableRequirementInterface* property), 140
- description (*ConstructableRequirementInterface* property), 142
- description (*Enumeration* property), 272
- description (*IntRequirement* property), 102
- description (*LayerListRequirement* property), 103
- description (*ListRequirement* property), 105
- description (*ModuleRequirement* property), 107
- description (*MultiRequirement* property), 108
- description (*PluginRequirement* property), 110
- description (*RequirementInterface* property), 145
- description (*SimpleTypeRequirement* property), 147
- description (*StringRequirement* property), 112
- description (*SymbolTableRequirement* property), 113
- description (*TranslationLayerRequirement* property), 115
- description (*URIRequirement* property), 116
- description (*VersionRequirement* property), 118
- DescStateEnum (class in *volatility3.plugins.linux.kmsg*), 579
- destroy() (*AVMLLayer* method), 182
- destroy() (*BufferDataLayer* method), 226
- destroy() (*DataLayerInterface* method), 154
- destroy() (*Elf64Layer* method), 192
- destroy() (*FileLayer* method), 228
- destroy() (*Intel* method), 195
- destroy() (*Intel32e* method), 198
- destroy() (*IntelPAE* method), 201
- destroy() (*LimeLayer* method), 215
- destroy() (*LinearlyMappedLayer* method), 217
- destroy() (*NonLinearlySegmentedLayer* method), 239
- destroy() (*PdbMSFStream* method), 220
- destroy() (*PdbMultiStreamFormat* method), 223
- destroy() (*QemuSuspendLayer* method), 231
- destroy() (*RegistryHive* method), 234
- destroy() (*SegmentedLayer* method), 241
- destroy() (*TranslationLayerInterface* method), 159
- destroy() (*VmwareLayer* method), 244
- destroy() (*WindowsCrashDump32Layer* method), 185
- destroy() (*WindowsCrashDump64Layer* method), 188
- destroy() (*WindowsIntel* method), 203
- destroy() (*WindowsIntel32e* method), 206
- destroy() (*WindowsIntelPAE* method), 209
- destroy() (*WindowsMixin* method), 212
- destroy() (*XenCoreDumpLayer* method), 247
- detach() (*NullFileHandler* method), 51
- determine_extended_value() (*PdbReader* method), 517
- determine_map() (*VirtMap* class method), 752
- determine_tcpip_version() (*NetScan* class method), 713
- determine_valid_kernel() (*KernelPDBScanner* method), 79
- DEVICE_OBJECT (class in *volatility3.framework.symbols.windows.extensions*), 435
- DEVICE_OBJECT.VolTemplateProxy (class in *volatility3.framework.symbols.windows.extensions*), 435
- DeviceTree (class in *volatility3.plugins.windows.devicetree*), 666
- devname (*MountInfoData* attribute), 592
- disassemble() (*Volshell* method), 53, 56, 59, 62
- Disassembly (class in *volatility3.framework.interfaces.renderers*), 169
- display_bytes() (*Volshell* method), 53, 56, 59, 62

- display_disassembly() (in module volatility3.cli.text_renderer), 66
- display_doublewords() (Volshell method), 53, 56, 59, 62
- display_plugin_output() (Volshell method), 53, 56, 59, 62
- display_quadwords() (Volshell method), 53, 56, 59, 62
- display_symbols() (Volshell method), 53, 56, 59, 62
- display_type() (Volshell method), 53, 56, 59, 62
- display_words() (Volshell method), 53, 56, 59, 62
- distro_re (QemuSuspendLayer attribute), 231
- DllList (class in volatility3.plugins.windows.dlllist), 667
- do_get_path() (LinuxUtilities class method), 324
- download_pdb_isf() (PDBUtility class method), 519
- DRIVER_OBJECT (class in volatility3.framework.symbols.windows.extensions), 437
- DRIVER_OBJECT.VolTemplateProxy (class in volatility3.framework.symbols.windows.extensions), 437
- DriverIrp (class in volatility3.plugins.windows.driverirp), 669
- DriverModule (class in volatility3.plugins.windows.drivermodule), 671
- DriverScan (class in volatility3.plugins.windows.driverscan), 673
- DtbSelfRef32bit (class in volatility3.framework.automagic.windows), 92
- DtbSelfRef64bit (class in volatility3.framework.automagic.windows), 92
- DtbSelfRef64bitOldWindows (class in volatility3.framework.automagic.windows), 92
- DtbSelfReferential (class in volatility3.framework.automagic.windows), 92
- DtbSelfRefPae (class in volatility3.framework.automagic.windows), 92
- DummyLock (class in volatility3.framework.layers.physical), 227
- DummyProgress (class in volatility3.framework.interfaces.layers), 156
- dump_file_producer() (DumpFiles class method), 675
- dump_header_name (WindowsCrashDump32Layer attribute), 186
- dump_header_name (WindowsCrashDump64Layer attribute), 188
- dump_pe() (DllList class method), 668
- DumpFiles (class in volatility3.plugins.windows.dumpfiles), 675
- dynamic_sections() (elf_phdr method), 394
- elf (class in volatility3.framework.symbols.linux.extensions.elf), 392
- elf.VolTemplateProxy (class in volatility3.framework.symbols.linux.extensions.elf), 392
- Elf64Layer (class in volatility3.framework.layers.elf), 191
- Elf64Stacker (class in volatility3.framework.layers.elf), 194
- ELF_CLASS (Elf64Layer attribute), 191
- ELF_CLASS (XenCoreDumpLayer attribute), 246
- elf_dump() (Elfs class method), 572
- elf_phdr (class in volatility3.framework.symbols.linux.extensions.elf), 393
- elf_phdr.VolTemplateProxy (class in volatility3.framework.symbols.linux.extensions.elf), 394
- elf_sym (class in volatility3.framework.symbols.linux.extensions.elf), 395
- elf_sym.VolTemplateProxy (class in volatility3.framework.symbols.linux.extensions.elf), 396
- ElfFormatException, 194
- Elfs (class in volatility3.plugins.linux.elfs), 571
- empty_lm (Hashdump attribute), 687
- empty_nt (Hashdump attribute), 687
- encode() (String method), 287
- endswith() (Bytes method), 260
- endswith() (HexBytes method), 308
- endswith() (MultiTypeData method), 313
- endswith() (String method), 287
- ENUM (SymbolType attribute), 319
- enumerate_capabilities() (kernel_cap_struct method), 340
- enumerate_capabilities() (kernel_cap_t method), 343
- enumerate_structures_by_port() (NetStat class method), 715
- Enumeration (class in volatility3.framework.objects), 270
- Enumeration.VolTemplateProxy (class in volatility3.framework.objects), 270
- enumerations (BaseSymbolTableInterface property), 172
- enumerations (BashIntermedSymbols property), 399
- enumerations (IntermediateSymbolTable property), 527
- enumerations (ISFormatTable property), 524
- enumerations (LinuxKernelIntermedSymbols property), 322
- enumerations (MacKernelIntermedSymbols property), 402
- enumerations (NativeTable property), 554
- enumerations (NativeTableInterface property), 174
- enumerations (SymbolTableInterface property), 178

- enumerations (*Version1Format* property), 530
 - enumerations (*Version2Format* property), 533
 - enumerations (*Version3Format* property), 536
 - enumerations (*Version4Format* property), 539
 - enumerations (*Version5Format* property), 542
 - enumerations (*Version6Format* property), 545
 - enumerations (*Version7Format* property), 548
 - enumerations (*Version8Format* property), 551
 - enumerations (*WindowsKernelIntermedSymbols* property), 430
 - Envvars (class in *volatility3.plugins.linux.envvars*), 573
 - Envvars (class in *volatility3.plugins.windows.envvars*), 677
 - environment variable
 - PYTHONPATH, 23
 - environment_variables() (*EPROCESS* method), 440
 - EPROCESS (class in *volatility3.framework.symbols.windows.extensions*), 439
 - EPROCESS.VolTemplateProxy (class in *volatility3.framework.symbols.windows.extensions*), 439
 - error() (*HelpfulArgParser* method), 67
 - ETHREAD (class in *volatility3.framework.symbols.windows.extensions*), 441
 - ETHREAD.VolTemplateProxy (class in *volatility3.framework.symbols.windows.extensions*), 442
 - euid (*cred* property), 330
 - euid (*TaskData* attribute), 563
 - event_types (*Kevents* attribute), 620
 - event_types (*VFSevents* attribute), 645
 - EX_FAST_REF (class in *volatility3.framework.symbols.windows.extensions*), 443
 - EX_FAST_REF.VolTemplateProxy (class in *volatility3.framework.symbols.windows.extensions*), 444
 - exclusion_list (*AutomagicInterface* attribute), 135
 - exclusion_list (*AVMLStacker* attribute), 184
 - exclusion_list (*ConstructionMagic* attribute), 71
 - exclusion_list (*Elf64Stacker* attribute), 194
 - exclusion_list (*KernelModule* attribute), 77
 - exclusion_list (*KernelPDBScanner* attribute), 79
 - exclusion_list (*LayerStacker* attribute), 83
 - exclusion_list (*LimeStacker* attribute), 216
 - exclusion_list (*LinuxIntelStacker* attribute), 72
 - exclusion_list (*LinuxSymbolFinder* attribute), 73
 - exclusion_list (*MacIntelStacker* attribute), 74
 - exclusion_list (*MacSymbolFinder* attribute), 75
 - exclusion_list (*QemuStacker* attribute), 230
 - exclusion_list (*StackerLayerInterface* attribute), 136
 - exclusion_list (*SymbolCacheMagic* attribute), 89
 - exclusion_list (*SymbolFinder* attribute), 91
 - exclusion_list (*VmwareStacker* attribute), 246
 - exclusion_list (*WindowsCrashDumpStacker* attribute), 191
 - exclusion_list (*WindowsIntelStacker* attribute), 95
 - exclusion_list (*WinSwapLayers* attribute), 93
 - exclusion_list (*XenCoreDumpStacker* attribute), 249
 - ExecutiveObject (class in *volatility3.framework.symbols.windows.extensions.pool*), 489
 - ExecutiveObject.VolTemplateProxy (class in *volatility3.framework.symbols.windows.extensions.pool*), 489
 - exit() (*HelpfulArgParser* method), 67
 - EXIT_DEAD (*DescExitStateEnum* attribute), 599
 - EXIT_TRACE (*DescExitStateEnum* attribute), 599
 - EXIT_ZOMBIE (*DescExitStateEnum* attribute), 599
 - expandtabs() (*Bytes* method), 260
 - expandtabs() (*HexBytes* method), 309
 - expandtabs() (*MultiTypeData* method), 313
 - expandtabs() (*String* method), 287
 - extended_flags (*vm_area_struct* attribute), 384
 - extract_data() (*QemuSuspendLayer* method), 231
- ## F
- FACILITIES (*ABCKmsg* attribute), 578
 - FACILITIES (*KmsgFiveTen* attribute), 582
 - FACILITIES (*KmsgLegacy* attribute), 583
 - fdel (classproperty attribute), 47
 - fget (classproperty attribute), 47
 - fields (*MountInfoData* attribute), 592
 - file_handler_class_factory() (*CommandLine* method), 48
 - file_handler_class_factory() (*VolShell* method), 49
 - file_name_with_device() (*FILE_OBJECT* method), 446
 - FILE_OBJECT (class in *volatility3.framework.symbols.windows.extensions*), 445
 - FILE_OBJECT.VolTemplateProxy (class in *volatility3.framework.symbols.windows.extensions*), 445
 - file_symbol_url() (*BashIntermedSymbols* class method), 399
 - file_symbol_url() (*IntermediateSymbolTable* class method), 527
 - file_symbol_url() (*LinuxKernelIntermedSymbols* class method), 322
 - file_symbol_url() (*MacKernelIntermedSymbols* class method), 402
 - file_symbol_url() (*WindowsKernelIntermedSymbols* class method), 430

`fileglob` (class in `volatility3.framework.symbols.mac.extensions`), 405

`fileglob.VolTemplateProxy` (class in `volatility3.framework.symbols.mac.extensions`), 405

`FileHandlerInterface` (class in `volatility3.framework.interfaces.plugins`), 165

`FileLayer` (class in `volatility3.framework.layers.physical`), 227

`fileno()` (`FileHandlerInterface` method), 165

`fileno()` (`NullFileHandler` method), 51

`FILEOFFSET_MASK` (VACB attribute), 469

`files_descriptors_for_process()` (`LinuxUtilities` class method), 325

`files_descriptors_for_process()` (`MacUtilities` class method), 404

`files_struct` (class in `volatility3.framework.symbols.linux.extensions`), 333

`files_struct.VolTemplateProxy` (class in `volatility3.framework.symbols.linux.extensions`), 334

`FileScan` (class in `volatility3.plugins.windows.filescan`), 678

`find()` (`Bytes` method), 260

`find()` (`HexBytes` method), 309

`find()` (`MultiTypeData` method), 313

`find()` (`String` method), 287

`find_aslr` (`SymbolFinder` attribute), 91

`find_aslr()` (`LinuxIntelStacker` class method), 72

`find_aslr()` (`LinuxSymbolFinder` method), 73

`find_aslr()` (`MacIntelStacker` class method), 74

`find_aslr()` (`MacSymbolFinder` class method), 75

`find_cookie()` (`Handles` class method), 684

`find_level()` (`PsTree` method), 601, 732

`find_location()` (`CacheManagerInterface` method), 85

`find_location()` (`SqliteCache` method), 87

`find_module()` (`WarningFindSpec` method), 47

`find_port_pools()` (`NetStat` class method), 716

`find_requirements()` (`AutomagicInterface` method), 135

`find_requirements()` (`ConstructionMagic` method), 71

`find_requirements()` (`KernelModule` method), 77

`find_requirements()` (`KernelPDBScanner` method), 79

`find_requirements()` (`LayerStacker` method), 83

`find_requirements()` (`LinuxSymbolFinder` method), 73

`find_requirements()` (`MacSymbolFinder` method), 75

`find_requirements()` (`SymbolCacheMagic` method), 89

`find_requirements()` (`SymbolFinder` method), 91

`find_requirements()` (`WinSwapLayers` method), 93

`find_sar_value()` (`Handles` method), 684

`find_session_layer()` (`ModScan` class method), 705

`find_session_layer()` (`Modules` class method), 708

`find_sid_re()` (in module `volatility3.plugins.windows.getsids`), 684

`find_spec()` (`WarningFindSpec` static method), 47

`find_suitable_requirements()` (`LayerStacker` class method), 83

`find_swap_requirement()` (`WinSwapLayers` static method), 94

`find_version_info()` (`VerInfo` class method), 750

`find_virtual_layers_from_req()` (`KernelPDBScanner` method), 80

`fix_image_base()` (`IMAGE_DOS_HEADER` method), 486

`Flags` (class in `volatility3.framework.symbols.wrappers`), 555

`Float` (class in `volatility3.framework.objects`), 273

`Float.VolTemplateProxy` (class in `volatility3.framework.objects`), 273

`flush()` (`FileHandlerInterface` method), 165

`flush()` (`NullFileHandler` method), 51

`format()` (`String` method), 287

`format_help()` (`HelpfulArgParser` method), 67

`format_map()` (`String` method), 288

`format_mapping` (`Version4Format` attribute), 539

`format_mapping` (`Version5Format` attribute), 542

`format_mapping` (`Version6Format` attribute), 545

`format_mapping` (`Version7Format` attribute), 548

`format_mapping` (`Version8Format` attribute), 551

`format_usage()` (`HelpfulArgParser` method), 68

`format_usage()` (`HelpfulSubparserAction` method), 68

`ForwardArrayCount` (class in `volatility3.framework.symbols.windows.pdbconv`), 517

`FrameworkInfo` (class in `volatility3.plugins.frameworkinfo`), 757

`FREE` (`PoolType` attribute), 722

`free_layer_name()` (`LayerContainer` method), 157

`free_module_name()` (`ModuleCollection` method), 130

`free_module_name()` (`ModuleContainer` method), 150

`free_table_name()` (`SymbolSpace` method), 318

`free_table_name()` (`SymbolSpaceInterface` method), 176

`from_bytes()` (`Bin` method), 306

`from_bytes()` (`BitField` method), 254

`from_bytes()` (`Boolean` method), 257

`from_bytes()` (`Char` method), 266

`from_bytes()` (`Enumeration` method), 272

`from_bytes()` (`Hex` method), 307

`from_bytes()` (`Integer` method), 279

`from_bytes()` (`Parallelism` method), 121

`from_bytes()` (`Pointer` method), 282

`from_bytes()` (`PoolType` method), 723

from_bytes() (*RegKeyFlags* method), 511
 from_bytes() (*TimeLinerType* method), 763
 fromhex() (*Bytes* method), 261
 fromhex() (*Float* method), 274
 fromhex() (*HexBytes* method), 309
 fromhex() (*MultiTypeData* method), 313
 fs_struct (class in *volatility3.framework.symbols.linux.extensions*), 335
 fs_struct.VolTemplateProxy (class in *volatility3.framework.symbols.linux.extensions*), 335
 fset (classproperty attribute), 47
 full_path() (*vnnode* method), 428
 Function (class in *volatility3.framework.objects*), 276
 Function.VolTemplateProxy (class in *volatility3.framework.objects*), 276

G

generate() (*WindowsIdentifier* class method), 90
 generate_kernel_handler_info() (*LinuxUtilities* class method), 325
 generate_kernel_handler_info() (*MacUtilities* class method), 404
 generate_mapping() (*Strings* class method), 738
 generate_pool_scan() (*PoolScanner* class method), 720
 generate_timeline() (*Bash* method), 559, 609
 generate_timeline() (*DILLList* method), 668
 generate_timeline() (*MFTScan* method), 704
 generate_timeline() (*NetScan* method), 713
 generate_timeline() (*NetStat* method), 716
 generate_timeline() (*PsList* method), 727
 generate_timeline() (*PsScan* method), 729
 generate_timeline() (*Sessions* method), 733
 generate_timeline() (*SymlinkScan* method), 742
 generate_timeline() (*TimeLinerInterface* method), 762
 generate_treegrid() (*Volshell* method), 54, 56, 59, 62
 generator() (*HierarchicalDict* method), 144
 GenericIntelProcess (class in *volatility3.framework.symbols.generic*), 319
 GenericIntelProcess.VolTemplateProxy (class in *volatility3.framework.symbols.generic*), 319
 get() (*HierarchicalDict* method), 144
 get() (*LayerContainer* method), 157
 get() (*ModuleCollection* method), 130
 get() (*ModuleContainer* method), 150
 get() (*ObjectInformation* method), 161
 get() (*ReadOnlyMapping* method), 163
 get() (*SymbolSpace* method), 318
 get() (*SymbolSpaceInterface* method), 176
 get_absolute_symbol_address() (*Configurable-Module* method), 123

get_absolute_symbol_address() (*Module* method), 128
 get_absolute_symbol_address() (*ModuleInterface* method), 151
 get_absolute_symbol_address() (*SizedModule* method), 131
 get_address() (*sockaddr* method), 417
 get_attached_devices() (*DEVICE_OBJECT* method), 436
 get_available_pages() (*CONTROL_AREA* method), 433
 get_available_pages() (*SHARED_CACHE_MAP* method), 463
 get_binary() (*SERVICE_RECORD* method), 515
 get_block_offset() (*HMAP_ENTRY* method), 509
 get_bootable_flag() (*PARTITION_ENTRY* method), 475
 get_bootkey() (*Hashdump* class method), 687
 get_buffer() (*SUMMARY_DUMP* method), 471
 get_buffer_char() (*SUMMARY_DUMP* method), 471
 get_buffer_long() (*SUMMARY_DUMP* method), 471
 get_build_lab() (*KDDEBUGGER_DATA64* method), 473
 get_caller() (*ABCKmsg* method), 578
 get_caller() (*KmsgFiveTen* method), 582
 get_caller() (*KmsgLegacy* method), 583
 get_caller_text() (*ABCKmsg* method), 578
 get_caller_text() (*KmsgFiveTen* method), 582
 get_caller_text() (*KmsgLegacy* method), 583
 get_capabilities() (*kernel_cap_struct* method), 340
 get_capabilities() (*kernel_cap_t* method), 343
 get_cell() (*RegistryHive* method), 234
 get_cmdline() (*CmdLine* class method), 663
 get_command() (*hist_entry* method), 390
 get_commit_charge() (*MMVAD* method), 456
 get_commit_charge() (*MMVAD_SHORT* method), 458
 get_connection_info() (*socket* method), 420
 get_converted_connection_info() (*socket* method), 421
 get_core_size() (*module* method), 355
 get_create_time() (*EPROCESS* method), 440
 get_create_time() (*OBJECT_SYMBOLIC_LINK* method), 461
 get_cross_thread_flags() (*ETHREAD* method), 442
 get_csdversion() (*KDDEBUGGER_DATA64* method), 473
 get_ctltype() (*sysctl_oid* method), 422
 get_default() (*HelpfulArgParser* method), 68
 get_dentry() (*struct_file* method), 372
 get_dentry_current() (*mount* method), 358
 get_dentry_current() (*vfsmount* method), 381
 get_dentry_parent() (*mount* method), 358
 get_dentry_parent() (*vfsmount* method), 381

`get_depends()` (*Info class method*), 689
`get_device_name()` (*DEVICE_OBJECT method*), 436
`get_devices()` (*DRIVER_OBJECT method*), 438
`get_devname()` (*mount method*), 358
`get_devname()` (*vfsmount method*), 381
`get_dict_lines()` (*KmsgFiveTen method*), 582
`get_dict_lines()` (*KmsgLegacy method*), 583
`get_disk_signature()` (*PARTITION_TABLE method*), 478
`get_display()` (*SERVICE_RECORD method*), 515
`get_dominating_id()` (*mount method*), 358
`get_driver_name()` (*DRIVER_OBJECT method*), 438
`get_dst_addr()` (*inet_sock method*), 338
`get_dst_port()` (*inet_sock method*), 338
`get_dst_portid()` (*netlink_sock method*), 363
`get_elf_table_name()` (*module method*), 355
`get_end()` (*MMVAD method*), 456
`get_end()` (*MMVAD_SHORT method*), 458
`get_ending_chs()` (*PARTITION_ENTRY method*), 475
`get_ending_cylinder()` (*PARTITION_ENTRY method*), 475
`get_enumeration()` (*BashIntermedSymbols method*), 399
`get_enumeration()` (*ConfigurableModule method*), 123
`get_enumeration()` (*IntermediateSymbolTable method*), 527
`get_enumeration()` (*LinuxKernelIntermedSymbols method*), 322
`get_enumeration()` (*MacKernelIntermedSymbols method*), 402
`get_enumeration()` (*Module method*), 128
`get_enumeration()` (*ModuleInterface method*), 151
`get_enumeration()` (*NativeTable method*), 554
`get_enumeration()` (*NativeTableInterface method*), 174
`get_enumeration()` (*SizedModule method*), 131
`get_enumeration()` (*SymbolSpace method*), 318
`get_enumeration()` (*SymbolSpaceInterface method*), 176
`get_enumeration()` (*Version1Format method*), 530
`get_enumeration()` (*Version2Format method*), 533
`get_enumeration()` (*Version3Format method*), 536
`get_enumeration()` (*Version4Format method*), 539
`get_enumeration()` (*Version5Format method*), 542
`get_enumeration()` (*Version6Format method*), 545
`get_enumeration()` (*Version7Format method*), 548
`get_enumeration()` (*Version8Format method*), 551
`get_enumeration()` (*WindowsKernelIntermedSymbols method*), 431
`get_exit_time()` (*EPROCESS method*), 440
`get_facility_text()` (*ABCKmsg class method*), 578
`get_facility_text()` (*KmsgFiveTen class method*), 582
`get_facility_text()` (*KmsgLegacy class method*), 583
`get_family()` (*inet_sock method*), 338
`get_family()` (*sock method*), 369
`get_family()` (*socket method*), 421
`get_fds()` (*files_struct method*), 334
`get_fg_type()` (*fileglob method*), 406
`get_file_name()` (*MMVAD method*), 456
`get_file_name()` (*MMVAD_SHORT method*), 458
`get_file_offset()` (*VACB method*), 469
`get_flags()` (*vm_area_struct method*), 385
`get_flags_access()` (*mount method*), 358
`get_flags_access()` (*super_block method*), 375
`get_flags_access()` (*vfsmount method*), 381
`get_flags_opts()` (*mount method*), 359
`get_flags_opts()` (*super_block method*), 375
`get_flags_opts()` (*vfsmount method*), 381
`get_full_key_name()` (*CM_KEY_BODY method*), 503
`get_full_name()` (*MFTFileName method*), 484
`get_guid_from_mz()` (*PDBUtility class method*), 519
`get_handle_count()` (*EPROCESS method*), 440
`get_hash()` (*CacheManagerInterface method*), 85
`get_hash()` (*MBRScan class method*), 699
`get_hash()` (*SqliteCache method*), 87
`get_hbootkey()` (*Hashdump class method*), 687
`get_header()` (*WindowsCrashDump32Layer method*), 186
`get_header()` (*WindowsCrashDump64Layer method*), 188
`get_hive_key()` (*Hashdump class method*), 687
`get_identifier()` (*CacheManagerInterface method*), 85
`get_identifier()` (*IdentifierProcessor class method*), 86
`get_identifier()` (*LinuxIdentifier class method*), 86
`get_identifier()` (*MacIdentifier class method*), 86
`get_identifier()` (*SqliteCache method*), 87
`get_identifier()` (*WindowsIdentifier class method*), 90
`get_identifier_dictionary()` (*CacheManagerInterface method*), 85
`get_identifier_dictionary()` (*SqliteCache method*), 87
`get_identifiers()` (*CacheManagerInterface method*), 85
`get_identifiers()` (*SqliteCache method*), 88
`get_init_size()` (*module method*), 355
`get_inode()` (*mnt_namespace method*), 353
`get_inode()` (*net method*), 361
`get_inode()` (*sock method*), 369
`get_inode()` (*socket method*), 371
`get_inode()` (*unix_sock method*), 379

get_inpcb() (*socket method*), 421
 get_ipv4_info() (*inpcb method*), 409
 get_ipv6_info() (*inpcb method*), 409
 get_is_wow64() (*EPROCESS method*), 440
 get_json() (*PdbReader method*), 518
 get_kdbg_structure() (*Info class method*), 690
 get_kernel_cap_full() (*kernel_cap_struct method*), 341
 get_kernel_cap_full() (*kernel_cap_t method*), 343
 get_kernel_module() (*Info class method*), 690
 get_key() (*POOL_TRACKER_BIG_PAGES method*), 499
 get_key() (*RegistryHive method*), 234
 get_key_path() (*CM_KEY_NODE method*), 505
 get_kset_modules() (*Check_modules class method*), 568
 get_kuser_structure() (*Info class method*), 690
 get_last_cap_value() (*kernel_cap_struct class method*), 341
 get_last_cap_value() (*kernel_cap_t class method*), 343
 get_left_child() (*MMVAD method*), 456
 get_left_child() (*MMVAD_SHORT method*), 459
 get_level_text() (*ABCKmsg class method*), 578
 get_level_text() (*KmsgFiveTen class method*), 582
 get_level_text() (*KmsgLegacy class method*), 583
 get_link_name() (*OBJECT_SYMBOLIC_LINK method*), 461
 get_list_tasks() (*PsList class method*), 636
 get_listeners() (*kauth_scope method*), 411
 get_local_locations() (*CacheManagerInterface method*), 86
 get_local_locations() (*SqliteCache method*), 88
 get_location_statistics() (*CacheManagerInterface method*), 86
 get_location_statistics() (*SqliteCache method*), 88
 get_log_lines() (*KmsgFiveTen method*), 582
 get_log_lines() (*KmsgLegacy method*), 584
 get_lsa_key() (*Lsadump class method*), 695
 get_map_iter() (*proc method*), 413
 get_map_object() (*vm_map_object method*), 426
 get_maple_tree_iter() (*mm_struct method*), 351
 get_max_fds() (*files_struct method*), 334
 get_mmap_iter() (*mm_struct method*), 352
 get_mnt_flags() (*mount method*), 359
 get_mnt_flags() (*vfsmount method*), 382
 get_mnt_mountpoint() (*mount method*), 359
 get_mnt_mountpoint() (*vfsmount method*), 382
 get_mnt_parent() (*mount method*), 359
 get_mnt_parent() (*vfsmount method*), 382
 get_mnt_root() (*mount method*), 359
 get_mnt_root() (*vfsmount method*), 382
 get_mnt_sb() (*mount method*), 359
 get_mnt_sb() (*vfsmount method*), 382
 get_module_base() (*module method*), 355
 get_module_core() (*module method*), 355
 get_module_from_volobj_type() (*LinuxUtilities class method*), 325
 get_module_init() (*module method*), 355
 get_module_symbols_by_absolute_location() (*ModuleCollection method*), 130
 get_module_wrapper() (*in module volatility3.framework.contexts*), 134
 get_modules_by_symbol_tables() (*ModuleCollection method*), 130
 get_modules_by_symbol_tables() (*ModuleContainer method*), 150
 get_mount_points() (*mnt_namespace method*), 353
 get_mountinfo() (*MountInfo class method*), 590
 get_name() (*CM_KEY_NODE method*), 505
 get_name() (*CM_KEY_VALUE method*), 507
 get_name() (*CMHIVE method*), 501
 get_name() (*elf_sym method*), 396
 get_name() (*KMUTANT method*), 448
 get_name() (*module method*), 355
 get_name() (*RegistryHive method*), 234
 get_name() (*SERVICE_RECORD method*), 515
 get_name() (*unix_sock method*), 379
 get_name() (*vm_area_struct method*), 385
 get_names_for_driver() (*DriverScan class method*), 673
 get_nlkm() (*Cachedump static method*), 658
 get_node() (*RegistryHive method*), 234
 get_nt_header() (*IMAGE_DOS_HEADER method*), 486
 get_nthead_struct() (*Info class method*), 690
 get_number_of_bytes() (*POOL_TRACKER_BIG_PAGES method*), 499
 get_object() (*POOL_HEADER method*), 494
 get_object() (*POOL_HEADER_VISTA method*), 496
 get_object() (*vm_map_entry method*), 424
 get_object_header() (*DEVICE_OBJECT method*), 436
 get_object_header() (*DRIVER_OBJECT method*), 438
 get_object_header() (*EPROCESS method*), 440
 get_object_header() (*ExecutiveObject method*), 490
 get_object_header() (*FILE_OBJECT method*), 446
 get_object_header() (*KMUTANT method*), 448
 get_object_header() (*OBJECT_SYMBOLIC_LINK method*), 461
 get_object_type() (*OBJECT_HEADER method*), 492
 get_offset() (*vm_map_entry method*), 425
 get_osversion() (*PsScan class method*), 729
 get_page_offset() (*vm_area_struct method*), 385
 get_parent() (*MMVAD method*), 456

`get_parent()` (*MMVAD_SHORT* method), 459
`get_parent_mount()` (*mount* method), 359
`get_partition_type()` (*PARTITION_ENTRY* method), 475
`get_path()` (*vm_map_entry* method), 425
`get_path_mnt()` (*LinuxUtilities* class method), 325
`get_peb()` (*EPROCESS* method), 440
`get_peer_under_root()` (*mount* method), 359
`get_perms()` (*sysctl_oid* method), 423
`get_perms()` (*vm_map_entry* method), 425
`get_physical_layer_name()` (*KernelPDBScanner* method), 80
`get_pid()` (*SERVICE_RECORD* method), 515
`get_pool_header_table()` (*PoolScanner* class method), 721
`get_pool_type()` (*POOL_TRACKER_BIG_PAGES* method), 499
`get_portid()` (*netlink_sock* method), 363
`get_prefix()` (*ABCKmsg* method), 578
`get_prefix()` (*KmsgFiveTen* method), 582
`get_prefix()` (*KmsgLegacy* method), 584
`get_private_memory()` (*MMVAD* method), 456
`get_private_memory()` (*MMVAD_SHORT* method), 459
`get_process_memory_sections()` (*proc* method), 413
`get_process_memory_sections()` (*task_struct* method), 377
`get_program_headers()` (*elf* method), 392
`get_protection()` (*MMVAD* method), 456
`get_protection()` (*MMVAD_SHORT* method), 459
`get_protection()` (*vm_area_struct* method), 385
`get_protocol()` (*bt_sock* method), 329
`get_protocol()` (*inet_sock* method), 338
`get_protocol()` (*netlink_sock* method), 363
`get_protocol()` (*packet_sock* method), 365
`get_protocol()` (*sock* method), 369
`get_protocol()` (*unix_sock* method), 379
`get_protocol()` (*vsock_sock* method), 387
`get_protocol()` (*xdp_sock* method), 388
`get_protocol_as_string()` (*socket* method), 421
`get_pte()` (*CONTROL_AREA* method), 433
`get_range_alias()` (*vm_map_entry* method), 425
`get_record_tuple()` (*SvcScan* static method), 741
`get_render_options()` (*CLIRenderer* method), 64
`get_render_options()` (*CSVRenderer* method), 64
`get_render_options()` (*JsonLinesRenderer* method), 65
`get_render_options()` (*JsonRenderer* method), 65
`get_render_options()` (*NoneRenderer* method), 65
`get_render_options()` (*PrettyTextRenderer* method), 66
`get_render_options()` (*QuickTextRenderer* method), 66
`get_render_options()` (*Renderer* method), 169
`get_requirements()` (*ADS* class method), 702
`get_requirements()` (*AutomagicInterface* class method), 135
`get_requirements()` (*AVMLLayer* class method), 182
`get_requirements()` (*Banners* class method), 754
`get_requirements()` (*Bash* class method), 559, 609
`get_requirements()` (*BashIntermedSymbols* class method), 399
`get_requirements()` (*BigPools* class method), 656
`get_requirements()` (*BufferDataLayer* class method), 226
`get_requirements()` (*Cachedump* class method), 658
`get_requirements()` (*Callbacks* class method), 660
`get_requirements()` (*Capabilities* class method), 561
`get_requirements()` (*Check_afinfo* class method), 563
`get_requirements()` (*Check_creds* class method), 565
`get_requirements()` (*Check_idt* class method), 567
`get_requirements()` (*Check_modules* class method), 568
`get_requirements()` (*Check_syscall* class method), 570, 610
`get_requirements()` (*Check_sysctl* class method), 612
`get_requirements()` (*Check_trap_table* class method), 614
`get_requirements()` (*CmdLine* class method), 663
`get_requirements()` (*ComplexListRequirement* class method), 100
`get_requirements()` (*ConfigurableInterface* class method), 139
`get_requirements()` (*ConfigurableModule* class method), 123
`get_requirements()` (*ConfigWriter* class method), 755
`get_requirements()` (*ConstructionMagic* class method), 71
`get_requirements()` (*Crashinfo* class method), 664
`get_requirements()` (*DataLayerInterface* class method), 154
`get_requirements()` (*DeviceTree* class method), 666
`get_requirements()` (*DllList* class method), 668
`get_requirements()` (*DriverIrp* class method), 670
`get_requirements()` (*DriverModule* class method), 671
`get_requirements()` (*DriverScan* class method), 673
`get_requirements()` (*DumpFiles* class method), 675
`get_requirements()` (*Elf64Layer* class method), 192
`get_requirements()` (*Elfs* class method), 572
`get_requirements()` (*Envvars* class method), 573, 677
`get_requirements()` (*FileLayer* class method), 228
`get_requirements()` (*FileScan* class method), 679
`get_requirements()` (*FrameworkInfo* class method), 757
`get_requirements()` (*GetServiceSIDs* class method), 681
`get_requirements()` (*GetSIDs* class method), 682

`get_requirements()` (*Handles class method*), 684
`get_requirements()` (*Hashdump class method*), 687
`get_requirements()` (*HiveList class method*), 648
`get_requirements()` (*HiveScan class method*), 650
`get_requirements()` (*Ifconfig class method*), 615
`get_requirements()` (*Info class method*), 690
`get_requirements()` (*Intel class method*), 195
`get_requirements()` (*Intel32e class method*), 198
`get_requirements()` (*IntelPAE class method*), 201
`get_requirements()` (*IntermediateSymbolTable class method*), 527
`get_requirements()` (*IOMem class method*), 575
`get_requirements()` (*IsfInfo class method*), 759
`get_requirements()` (*ISFormatTable class method*), 524
`get_requirements()` (*JobLinks class method*), 692
`get_requirements()` (*Kauth_listeners class method*), 617
`get_requirements()` (*Kauth_scopes class method*), 618
`get_requirements()` (*KernelModule class method*), 77
`get_requirements()` (*KernelPDBScanner class method*), 80
`get_requirements()` (*Kevents class method*), 620
`get_requirements()` (*Keyboard_notifiers class method*), 577
`get_requirements()` (*Kmsg class method*), 580
`get_requirements()` (*LayerListRequirement class method*), 103
`get_requirements()` (*LayerStacker class method*), 83
`get_requirements()` (*LayerWriter class method*), 760
`get_requirements()` (*LdrModules class method*), 693
`get_requirements()` (*LimeLayer class method*), 215
`get_requirements()` (*LinearlyMappedLayer class method*), 218
`get_requirements()` (*LinuxKernelIntermedSymbols class method*), 322
`get_requirements()` (*LinuxSymbolFinder class method*), 73
`get_requirements()` (*List_Files class method*), 622
`get_requirements()` (*Lsadbump class method*), 695
`get_requirements()` (*Lsmod class method*), 585, 624
`get_requirements()` (*Lsof class method*), 587, 626
`get_requirements()` (*MacKernelIntermedSymbols class method*), 402
`get_requirements()` (*MacSymbolFinder class method*), 76
`get_requirements()` (*Malfind class method*), 589, 627, 697
`get_requirements()` (*Maps class method*), 593, 632
`get_requirements()` (*MBRScan class method*), 699
`get_requirements()` (*Memmap class method*), 700
`get_requirements()` (*MFTScan class method*), 704
`get_requirements()` (*ModScan class method*), 706
`get_requirements()` (*Module class method*), 128
`get_requirements()` (*ModuleInterface class method*), 151
`get_requirements()` (*ModuleRequirement class method*), 107
`get_requirements()` (*Modules class method*), 708
`get_requirements()` (*Mount class method*), 629
`get_requirements()` (*MountInfo class method*), 590
`get_requirements()` (*MutantScan class method*), 710
`get_requirements()` (*NetScan class method*), 713
`get_requirements()` (*NetStat class method*), 716
`get_requirements()` (*Netstat class method*), 630
`get_requirements()` (*NonLinearlySegmentedLayer class method*), 239
`get_requirements()` (*PdbMSFStream class method*), 220
`get_requirements()` (*PdbMultiStreamFormat class method*), 223
`get_requirements()` (*PluginInterface class method*), 167
`get_requirements()` (*PoolScanner class method*), 721
`get_requirements()` (*PrintKey class method*), 652
`get_requirements()` (*Privs class method*), 724
`get_requirements()` (*PsAux class method*), 595
`get_requirements()` (*Psaux class method*), 634
`get_requirements()` (*PsList class method*), 597, 636, 727
`get_requirements()` (*PsScan class method*), 599, 729
`get_requirements()` (*PsTree class method*), 601, 639, 732
`get_requirements()` (*QemuSuspendLayer class method*), 231
`get_requirements()` (*RegistryHive class method*), 235
`get_requirements()` (*SegmentedLayer class method*), 241
`get_requirements()` (*Sessions class method*), 733
`get_requirements()` (*SizedModule class method*), 131
`get_requirements()` (*Skeleton_Key_Check class method*), 735
`get_requirements()` (*Socket_filters class method*), 641
`get_requirements()` (*Sockstat class method*), 603
`get_requirements()` (*SSDT class method*), 737
`get_requirements()` (*Strings class method*), 739
`get_requirements()` (*SvcScan class method*), 741
`get_requirements()` (*SymbolCacheMagic class method*), 89
`get_requirements()` (*SymbolFinder class method*), 91
`get_requirements()` (*SymbolTableInterface class method*), 178
`get_requirements()` (*SymlinkScan class method*), 742
`get_requirements()` (*Timeliner class method*), 764
`get_requirements()` (*Timers class method*), 642
`get_requirements()` (*TranslationLayerInterface class method*), 159

`get_requirements()` (*Trustedbsd class method*), 644
`get_requirements()` (*tty_check class method*), 605
`get_requirements()` (*UserAssist class method*), 654
`get_requirements()` (*VadInfo class method*), 744
`get_requirements()` (*VadWalk class method*), 747
`get_requirements()` (*VadYaraScan class method*), 748
`get_requirements()` (*VerInfo class method*), 750
`get_requirements()` (*Version1Format class method*), 530
`get_requirements()` (*Version2Format class method*), 533
`get_requirements()` (*Version3Format class method*), 536
`get_requirements()` (*Version4Format class method*), 539
`get_requirements()` (*Version5Format class method*), 542
`get_requirements()` (*Version6Format class method*), 545
`get_requirements()` (*Version7Format class method*), 548
`get_requirements()` (*Version8Format class method*), 551
`get_requirements()` (*VFSevents class method*), 645
`get_requirements()` (*VirtMap class method*), 752
`get_requirements()` (*VmaYaraScan class method*), 607
`get_requirements()` (*VmwareLayer class method*), 244
`get_requirements()` (*Volshell class method*), 54, 56, 59, 62
`get_requirements()` (*WindowsCrashDump32Layer class method*), 186
`get_requirements()` (*WindowsCrashDump64Layer class method*), 189
`get_requirements()` (*WindowsIntel class method*), 204
`get_requirements()` (*WindowsIntel32e class method*), 206
`get_requirements()` (*WindowsIntelPAE class method*), 209
`get_requirements()` (*WindowsKernelIntermedSymbols class method*), 431
`get_requirements()` (*WindowsMixin class method*), 212
`get_requirements()` (*WinSwapLayers class method*), 94
`get_requirements()` (*XenCoreDumpLayer class method*), 247
`get_requirements()` (*YaraScan class method*), 766
`get_resident_filecontent()` (*MFTAttribute method*), 480
`get_resident_filename()` (*MFTAttribute method*), 480
`get_right_child()` (*MMVAD method*), 456
`get_right_child()` (*MMVAD_SHORT method*), 459
`get_root_dentry()` (*fs_struct method*), 336
`get_root_mnt()` (*fs_struct method*), 336
`get_secret_by_name()` (*Lsadump class method*), 695
`get_section_headers()` (*elf method*), 393
`get_sections()` (*IMAGE_NT_HEADERS method*), 488
`get_sections()` (*module method*), 355
`get_session_id()` (*EPROCESS method*), 440
`get_session_layers()` (*ModScan class method*), 706
`get_session_layers()` (*Modules class method*), 708
`get_sids()` (*TOKEN method*), 465
`get_signature()` (*MFTEntry method*), 482
`get_size()` (*MMVAD method*), 456
`get_size()` (*MMVAD_SHORT method*), 459
`get_size_from_index()` (*PdbReader method*), 518
`get_size_in_sectors()` (*PARTITION_ENTRY method*), 476
`get_slot_iter()` (*maple_tree method*), 350
`get_special_path()` (*vm_map_entry method*), 425
`get_src_addr()` (*inet_sock method*), 338
`get_src_port()` (*inet_sock method*), 338
`get_start()` (*MMVAD method*), 456
`get_start()` (*MMVAD_SHORT method*), 459
`get_starting_chs()` (*PARTITION_ENTRY method*), 476
`get_starting_cylinder()` (*PARTITION_ENTRY method*), 476
`get_starting_lba()` (*PARTITION_ENTRY method*), 476
`get_starting_sector()` (*PARTITION_ENTRY method*), 476
`get_state()` (*bt_sock method*), 329
`get_state()` (*inet_sock method*), 338
`get_state()` (*KTHREAD method*), 452
`get_state()` (*netlink_sock method*), 363
`get_state()` (*packet_sock method*), 365
`get_state()` (*sock method*), 369
`get_state()` (*socket method*), 371, 421
`get_state()` (*unix_sock method*), 379
`get_state()` (*vsock_sock method*), 387
`get_state()` (*xdp_sock method*), 389
`get_stream()` (*PdbMultiStreamFormat method*), 223
`get_string()` (*ABCKmsg method*), 579
`get_string()` (*KmsgFiveTen method*), 582
`get_string()` (*KmsgLegacy method*), 584
`get_string()` (*UNICODE_STRING method*), 467
`get_subkeys()` (*CM_KEY_NODE method*), 505
`get_subsection()` (*CONTROL_AREA method*), 434
`get_summary_header()` (*WindowsCrashDump32Layer method*), 186
`get_summary_header()` (*WindowsCrashDump64Layer method*), 189
`get_symbol()` (*BaseSymbolTableInterface method*), 172
`get_symbol()` (*BashIntermedSymbols method*), 399

- [get_symbol\(\)](#) (*ConfigurableModule method*), 123
[get_symbol\(\)](#) (*IntermediateSymbolTable method*), 528
[get_symbol\(\)](#) (*ISFormatTable method*), 524
[get_symbol\(\)](#) (*LinuxKernelIntermedSymbols method*), 322
[get_symbol\(\)](#) (*MacKernelIntermedSymbols method*), 402
[get_symbol\(\)](#) (*Module method*), 128
[get_symbol\(\)](#) (*module method*), 355
[get_symbol\(\)](#) (*ModuleInterface method*), 151
[get_symbol\(\)](#) (*NativeTable method*), 554
[get_symbol\(\)](#) (*NativeTableInterface method*), 174
[get_symbol\(\)](#) (*SizedModule method*), 132
[get_symbol\(\)](#) (*SymbolSpace method*), 318
[get_symbol\(\)](#) (*SymbolSpaceInterface method*), 176
[get_symbol\(\)](#) (*SymbolTableInterface method*), 178
[get_symbol\(\)](#) (*Version1Format method*), 530
[get_symbol\(\)](#) (*Version2Format method*), 533
[get_symbol\(\)](#) (*Version3Format method*), 536
[get_symbol\(\)](#) (*Version4Format method*), 539
[get_symbol\(\)](#) (*Version5Format method*), 542
[get_symbol\(\)](#) (*Version6Format method*), 545
[get_symbol\(\)](#) (*Version7Format method*), 548
[get_symbol\(\)](#) (*Version8Format method*), 551
[get_symbol\(\)](#) (*WindowsKernelIntermedSymbols method*), 431
[get_symbol_by_address\(\)](#) (*module method*), 356
[get_symbol_table_name\(\)](#) (*AggregateType method*), 250
[get_symbol_table_name\(\)](#) (*Array method*), 252
[get_symbol_table_name\(\)](#) (*BitField method*), 255
[get_symbol_table_name\(\)](#) (*Boolean method*), 258
[get_symbol_table_name\(\)](#) (*bpf_prog method*), 327
[get_symbol_table_name\(\)](#) (*bt_sock method*), 329
[get_symbol_table_name\(\)](#) (*Bytes method*), 261
[get_symbol_table_name\(\)](#) (*Char method*), 266
[get_symbol_table_name\(\)](#) (*ClassType method*), 269
[get_symbol_table_name\(\)](#) (*CM_KEY_BODY method*), 503
[get_symbol_table_name\(\)](#) (*CM_KEY_NODE method*), 505
[get_symbol_table_name\(\)](#) (*CM_KEY_VALUE method*), 507
[get_symbol_table_name\(\)](#) (*CMHIVE method*), 501
[get_symbol_table_name\(\)](#) (*CONTROL_AREA method*), 434
[get_symbol_table_name\(\)](#) (*cred method*), 331
[get_symbol_table_name\(\)](#) (*dentry method*), 332
[get_symbol_table_name\(\)](#) (*DEVICE_OBJECT method*), 436
[get_symbol_table_name\(\)](#) (*DRIVER_OBJECT method*), 438
[get_symbol_table_name\(\)](#) (*elf method*), 393
[get_symbol_table_name\(\)](#) (*elf_phdr method*), 394
[get_symbol_table_name\(\)](#) (*elf_sym method*), 396
[get_symbol_table_name\(\)](#) (*Enumeration method*), 272
[get_symbol_table_name\(\)](#) (*EPROCESS method*), 440
[get_symbol_table_name\(\)](#) (*ETHREAD method*), 442
[get_symbol_table_name\(\)](#) (*EX_FAST_REF method*), 444
[get_symbol_table_name\(\)](#) (*ExecutiveObject method*), 490
[get_symbol_table_name\(\)](#) (*FILE_OBJECT method*), 446
[get_symbol_table_name\(\)](#) (*fileglob method*), 406
[get_symbol_table_name\(\)](#) (*files_struct method*), 334
[get_symbol_table_name\(\)](#) (*Float method*), 275
[get_symbol_table_name\(\)](#) (*fs_struct method*), 336
[get_symbol_table_name\(\)](#) (*Function method*), 277
[get_symbol_table_name\(\)](#) (*GenericIntelProcess method*), 320
[get_symbol_table_name\(\)](#) (*hist_entry method*), 391
[get_symbol_table_name\(\)](#) (*HMAP_ENTRY method*), 509
[get_symbol_table_name\(\)](#) (*ifnet method*), 407
[get_symbol_table_name\(\)](#) (*IMAGE_DOS_HEADER method*), 486
[get_symbol_table_name\(\)](#) (*IMAGE_NT_HEADERS method*), 488
[get_symbol_table_name\(\)](#) (*inet_sock method*), 338
[get_symbol_table_name\(\)](#) (*inpcb method*), 409
[get_symbol_table_name\(\)](#) (*Integer method*), 279
[get_symbol_table_name\(\)](#) (*kauth_scope method*), 411
[get_symbol_table_name\(\)](#) (*KDDEBUG-GER_DATA64 method*), 473
[get_symbol_table_name\(\)](#) (*kernel_cap_struct method*), 341
[get_symbol_table_name\(\)](#) (*kernel_cap_t method*), 344
[get_symbol_table_name\(\)](#) (*KMUTANT method*), 448
[get_symbol_table_name\(\)](#) (*kobject method*), 345
[get_symbol_table_name\(\)](#) (*KSYSTEM_TIME method*), 450
[get_symbol_table_name\(\)](#) (*KTHREAD method*), 452
[get_symbol_table_name\(\)](#) (*LIST_ENTRY method*), 454
[get_symbol_table_name\(\)](#) (*list_head method*), 347
[get_symbol_table_name\(\)](#) (*maple_tree method*), 350
[get_symbol_table_name\(\)](#) (*MFTAttribute method*), 480
[get_symbol_table_name\(\)](#) (*MFTEntry method*), 482
[get_symbol_table_name\(\)](#) (*MFTFileName method*), 484
[get_symbol_table_name\(\)](#) (*mm_struct method*), 352
[get_symbol_table_name\(\)](#) (*MMVAD method*), 456
[get_symbol_table_name\(\)](#) (*MMVAD_SHORT method*), 459

`get_symbol_table_name()` (*mnt_namespace method*), 353

`get_symbol_table_name()` (*module method*), 356

`get_symbol_table_name()` (*mount method*), 359

`get_symbol_table_name()` (*net method*), 361

`get_symbol_table_name()` (*netlink_sock method*), 363

`get_symbol_table_name()` (*OBJECT_HEADER method*), 492

`get_symbol_table_name()` (*OBJECT_SYMBOLIC_LINK method*), 461

`get_symbol_table_name()` (*ObjectInterface method*), 162

`get_symbol_table_name()` (*packet_sock method*), 365

`get_symbol_table_name()` (*PARTITION_ENTRY method*), 476

`get_symbol_table_name()` (*PARTITION_TABLE method*), 478

`get_symbol_table_name()` (*Pointer method*), 283

`get_symbol_table_name()` (*POOL_HEADER method*), 495

`get_symbol_table_name()` (*POOL_HEADER_VISTA method*), 497

`get_symbol_table_name()` (*POOL_TRACKER_BIG_PAGES method*), 499

`get_symbol_table_name()` (*PrimitiveObject method*), 285

`get_symbol_table_name()` (*proc method*), 413

`get_symbol_table_name()` (*qstr method*), 367

`get_symbol_table_name()` (*queue_entry method*), 415

`get_symbol_table_name()` (*SERVICE_HEADER method*), 513

`get_symbol_table_name()` (*SERVICE_RECORD method*), 516

`get_symbol_table_name()` (*SHARED_CACHE_MAP method*), 463

`get_symbol_table_name()` (*sock method*), 369

`get_symbol_table_name()` (*sockaddr method*), 417

`get_symbol_table_name()` (*sockaddr_dl method*), 419

`get_symbol_table_name()` (*socket method*), 371, 421

`get_symbol_table_name()` (*String method*), 288

`get_symbol_table_name()` (*struct_file method*), 372

`get_symbol_table_name()` (*StructType method*), 293

`get_symbol_table_name()` (*SUMMARY_DUMP method*), 471

`get_symbol_table_name()` (*super_block method*), 375

`get_symbol_table_name()` (*sysctl_oid method*), 423

`get_symbol_table_name()` (*task_struct method*), 377

`get_symbol_table_name()` (*TOKEN method*), 466

`get_symbol_table_name()` (*UNICODE_STRING method*), 468

`get_symbol_table_name()` (*UnionType method*), 295

`get_symbol_table_name()` (*unix_sock method*), 379

`get_symbol_table_name()` (*VACB method*), 469

`get_symbol_table_name()` (*vfsmount method*), 382

`get_symbol_table_name()` (*vm_area_struct method*), 385

`get_symbol_table_name()` (*vm_map_entry method*), 425

`get_symbol_table_name()` (*vm_map_object method*), 426

`get_symbol_table_name()` (*vnode method*), 428

`get_symbol_table_name()` (*Void method*), 297

`get_symbol_table_name()` (*vsock_sock method*), 387

`get_symbol_table_name()` (*xdp_sock method*), 389

`get_symbol_type()` (*BaseSymbolTableInterface method*), 172

`get_symbol_type()` (*BashIntermedSymbols method*), 399

`get_symbol_type()` (*IntermediateSymbolTable method*), 528

`get_symbol_type()` (*ISFormatTable method*), 524

`get_symbol_type()` (*LinuxKernelIntermedSymbols method*), 323

`get_symbol_type()` (*MacKernelIntermedSymbols method*), 402

`get_symbol_type()` (*NativeTable method*), 554

`get_symbol_type()` (*NativeTableInterface method*), 174

`get_symbol_type()` (*SymbolTableInterface method*), 178

`get_symbol_type()` (*Version1Format method*), 530

`get_symbol_type()` (*Version2Format method*), 533

`get_symbol_type()` (*Version3Format method*), 536

`get_symbol_type()` (*Version4Format method*), 539

`get_symbol_type()` (*Version5Format method*), 542

`get_symbol_type()` (*Version6Format method*), 545

`get_symbol_type()` (*Version7Format method*), 548

`get_symbol_type()` (*Version8Format method*), 551

`get_symbol_type()` (*WindowsKernelIntermedSymbols method*), 431

`get_symbols()` (*elf method*), 393

`get_symbols()` (*module method*), 356

`get_symbols_by_absolute_location()` (*ConfigurableModule method*), 124

`get_symbols_by_absolute_location()` (*Module method*), 128

`get_symbols_by_absolute_location()` (*ModuleInterface method*), 151

`get_symbols_by_absolute_location()` (*SizedModule method*), 132

`get_symbols_by_location()` (*BaseSymbolTableInterface method*), 173

`get_symbols_by_location()` (*BashIntermedSymbols method*), 399

`get_symbols_by_location()` (*IntermediateSymbolTable method*), 528

`get_symbols_by_location()` (*ISFormatTable*

- method*), 524
- `get_symbols_by_location()` (*LinuxKernelInter-medSymbols method*), 323
- `get_symbols_by_location()` (*MacKernelInter-medSymbols method*), 402
- `get_symbols_by_location()` (*NativeTable method*), 554
- `get_symbols_by_location()` (*NativeTableInterface method*), 174
- `get_symbols_by_location()` (*SymbolSpace method*), 318
- `get_symbols_by_location()` (*SymbolSpaceInterface method*), 176
- `get_symbols_by_location()` (*SymbolTableInterface method*), 178
- `get_symbols_by_location()` (*Version1Format method*), 530
- `get_symbols_by_location()` (*Version2Format method*), 533
- `get_symbols_by_location()` (*Version3Format method*), 536
- `get_symbols_by_location()` (*Version4Format method*), 539
- `get_symbols_by_location()` (*Version5Format method*), 542
- `get_symbols_by_location()` (*Version6Format method*), 545
- `get_symbols_by_location()` (*Version7Format method*), 548
- `get_symbols_by_location()` (*Version8Format method*), 551
- `get_symbols_by_location()` (*WindowsKernelInter-medSymbols method*), 431
- `get_symbols_by_type()` (*BaseSymbolTableInterface method*), 173
- `get_symbols_by_type()` (*BashIntermedSymbols method*), 399
- `get_symbols_by_type()` (*IntermediateSymbolTable method*), 528
- `get_symbols_by_type()` (*ISFormatTable method*), 524
- `get_symbols_by_type()` (*LinuxKernelIntermedSymbols method*), 323
- `get_symbols_by_type()` (*MacKernelIntermedSymbols method*), 402
- `get_symbols_by_type()` (*NativeTable method*), 554
- `get_symbols_by_type()` (*NativeTableInterface method*), 175
- `get_symbols_by_type()` (*SymbolSpace method*), 318
- `get_symbols_by_type()` (*SymbolSpaceInterface method*), 177
- `get_symbols_by_type()` (*SymbolTableInterface method*), 179
- `get_symbols_by_type()` (*Version1Format method*), 531
- `get_symbols_by_type()` (*Version2Format method*), 533
- `get_symbols_by_type()` (*Version3Format method*), 536
- `get_symbols_by_type()` (*Version4Format method*), 539
- `get_symbols_by_type()` (*Version5Format method*), 542
- `get_symbols_by_type()` (*Version6Format method*), 545
- `get_symbols_by_type()` (*Version7Format method*), 548
- `get_symbols_by_type()` (*Version8Format method*), 551
- `get_symbols_by_type()` (*WindowsKernelInter-medSymbols method*), 431
- `get_symbols_names_and_addresses()` (*module method*), 356
- `get_tag()` (*MMVAD method*), 457
- `get_tag()` (*MMVAD_SHORT method*), 459
- `get_task()` (*proc method*), 413
- `get_task_capabilities()` (*Capabilities class method*), 561
- `get_task_fields()` (*PsList class method*), 597
- `get_tasks_capabilities()` (*Capabilities class method*), 561
- `get_tcp_state()` (*inpcb method*), 409
- `get_tcpip_module()` (*NetStat class method*), 716
- `get_text_from_data_ring()` (*KmsgFiveTen method*), 582
- `get_text_from_printk_log()` (*KmsgLegacy method*), 584
- `get_threads()` (*task_struct method*), 377
- `get_time()` (*KSYSTEM_TIME method*), 450
- `get_time_as_integer()` (*hist_entry method*), 391
- `get_time_object()` (*hist_entry method*), 391
- `get_timestamp_in_sec_str()` (*ABCKmsg method*), 579
- `get_timestamp_in_sec_str()` (*KmsgFiveTen method*), 582
- `get_timestamp_in_sec_str()` (*KmsgLegacy method*), 584
- `get_type()` (*BaseSymbolTableInterface method*), 173
- `get_type()` (*BashIntermedSymbols method*), 399
- `get_type()` (*bpf_prog method*), 327
- `get_type()` (*ConfigurableModule method*), 124
- `get_type()` (*IntermediateSymbolTable method*), 528
- `get_type()` (*ISFormatTable method*), 524
- `get_type()` (*LinuxKernelIntermedSymbols method*), 323
- `get_type()` (*MacKernelIntermedSymbols method*), 403
- `get_type()` (*Module method*), 128
- `get_type()` (*ModuleInterface method*), 151
- `get_type()` (*NativeTable method*), 554

`get_type()` (*NativeTableInterface* method), 175
`get_type()` (*SERVICE_RECORD* method), 516
`get_type()` (*SizedModule* method), 132
`get_type()` (*sock* method), 369
`get_type()` (*super_block* method), 375
`get_type()` (*SymbolSpace* method), 318
`get_type()` (*SymbolSpaceInterface* method), 177
`get_type()` (*SymbolTableInterface* method), 179
`get_type()` (*Version1Format* method), 531
`get_type()` (*Version2Format* method), 534
`get_type()` (*Version3Format* method), 536
`get_type()` (*Version4Format* method), 539
`get_type()` (*Version5Format* method), 542
`get_type()` (*Version6Format* method), 545
`get_type()` (*Version7Format* method), 548
`get_type()` (*Version8Format* method), 551
`get_type()` (*WindowsKernelIntermedSymbols* method), 431
`get_type_class()` (*BaseSymbolTableInterface* method), 173
`get_type_class()` (*BashIntermedSymbols* method), 399
`get_type_class()` (*IntermediateSymbolTable* method), 528
`get_type_class()` (*ISFormatTable* method), 524
`get_type_class()` (*LinuxKernelIntermedSymbols* method), 323
`get_type_class()` (*MacKernelIntermedSymbols* method), 403
`get_type_class()` (*NativeTable* method), 554
`get_type_class()` (*NativeTableInterface* method), 175
`get_type_class()` (*SymbolTableInterface* method), 179
`get_type_class()` (*Version1Format* method), 531
`get_type_class()` (*Version2Format* method), 534
`get_type_class()` (*Version3Format* method), 536
`get_type_class()` (*Version4Format* method), 540
`get_type_class()` (*Version5Format* method), 543
`get_type_class()` (*Version6Format* method), 546
`get_type_class()` (*Version7Format* method), 549
`get_type_class()` (*Version8Format* method), 552
`get_type_class()` (*WindowsKernelIntermedSymbols* method), 431
`get_type_from_index()` (*PdbReader* method), 518
`get_type_map()` (*Handles* class method), 684
`get_usable_plugins()` (*Timeliner* class method), 764
`get_user_hashes()` (*Hashdump* class method), 687
`get_user_keys()` (*Hashdump* class method), 687
`get_user_name()` (*Hashdump* class method), 687
`get_vad_maps()` (*VadYaraScan* static method), 748
`get_vad_root()` (*EPROCESS* method), 440
`get_vaddr()` (*elf_phdr* method), 395
`get_values()` (*CM_KEY_NODE* method), 505

`get_version_information()` (*VerInfo* class method), 750
`get_version_structure()` (*Info* class method), 690
`get_vfsmnt()` (*struct_file* method), 373
`get_vfsmnt_current()` (*mount* method), 359
`get_vfsmnt_current()` (*vfsmount* method), 382
`get_vfsmnt_parent()` (*mount* method), 359
`get_vfsmnt_parent()` (*vfsmount* method), 382
`get_vma_iter()` (*mm_struct* method), 352
`get_vma_maps()` (*VmaYaraScan* static method), 607
`get_vnode()` (*vm_map_entry* method), 425
`get_volatile()` (*CM_KEY_NODE* method), 506
`get_wait_reason()` (*KTHREAD* method), 452
`get_wow_64_process()` (*EPROCESS* method), 440
`get_yarascan_option_requirements()` (*YaraScan* class method), 766
`getbuffer()` (*NullFileHandler* method), 51
`GetServiceSIDs` (class in *volatility3.plugins.windows.getservicesids*), 680
`GetSIDs` (class in *volatility3.plugins.windows.getsids*), 682
`getter()` (*classproperty* method), 47
`getvalue()` (*NullFileHandler* method), 51
`group_structure` (*VmwareLayer* attribute), 244

H

`handler_order` (*JarHandler* attribute), 237
`handler_order` (*OfflineHandler* attribute), 237
`handler_order` (*VolatilityHandler* attribute), 238
`Handles` (class in *volatility3.plugins.windows.handles*), 684
`handles()` (*Handles* method), 685
`has_capability()` (*kernel_cap_struct* method), 341
`has_capability()` (*kernel_cap_t* method), 344
`has_enumeration()` (*ConfigurableModule* method), 124
`has_enumeration()` (*Module* method), 128
`has_enumeration()` (*ModuleInterface* method), 151
`has_enumeration()` (*SizedModule* method), 132
`has_enumeration()` (*SymbolSpace* method), 318
`has_enumeration()` (*SymbolSpaceInterface* method), 177
`has_member()` (*AggregateType* method), 250
`has_member()` (*AggregateType.VolTemplateProxy* class method), 250
`has_member()` (*Array* method), 252
`has_member()` (*Array.VolTemplateProxy* class method), 251
`has_member()` (*BitField* method), 255
`has_member()` (*BitField.VolTemplateProxy* class method), 253
`has_member()` (*Boolean* method), 258
`has_member()` (*Boolean.VolTemplateProxy* class method), 256

[has_member\(\)](#) (*bpf_prog* method), 327
[has_member\(\)](#) (*bpf_prog.VolTemplateProxy* class method), 326
[has_member\(\)](#) (*bt_sock* method), 329
[has_member\(\)](#) (*bt_sock.VolTemplateProxy* class method), 328
[has_member\(\)](#) (*Bytes* method), 261
[has_member\(\)](#) (*Bytes.VolTemplateProxy* class method), 259
[has_member\(\)](#) (*Char* method), 267
[has_member\(\)](#) (*Char.VolTemplateProxy* class method), 265
[has_member\(\)](#) (*ClassType* method), 269
[has_member\(\)](#) (*ClassType.VolTemplateProxy* class method), 268
[has_member\(\)](#) (*CM_KEY_BODY* method), 503
[has_member\(\)](#) (*CM_KEY_BODY.VolTemplateProxy* class method), 503
[has_member\(\)](#) (*CM_KEY_NODE* method), 506
[has_member\(\)](#) (*CM_KEY_NODE.VolTemplateProxy* class method), 504
[has_member\(\)](#) (*CM_KEY_VALUE* method), 508
[has_member\(\)](#) (*CM_KEY_VALUE.VolTemplateProxy* class method), 507
[has_member\(\)](#) (*CMHIVE* method), 501
[has_member\(\)](#) (*CMHIVE.VolTemplateProxy* class method), 501
[has_member\(\)](#) (*CONTROL_AREA* method), 434
[has_member\(\)](#) (*CONTROL_AREA.VolTemplateProxy* class method), 433
[has_member\(\)](#) (*cred* method), 331
[has_member\(\)](#) (*cred.VolTemplateProxy* class method), 330
[has_member\(\)](#) (*dentry* method), 333
[has_member\(\)](#) (*dentry.VolTemplateProxy* class method), 332
[has_member\(\)](#) (*DEVICE_OBJECT* method), 436
[has_member\(\)](#) (*DEVICE_OBJECT.VolTemplateProxy* class method), 435
[has_member\(\)](#) (*DRIVER_OBJECT* method), 438
[has_member\(\)](#) (*DRIVER_OBJECT.VolTemplateProxy* class method), 437
[has_member\(\)](#) (*elf* method), 393
[has_member\(\)](#) (*elf.VolTemplateProxy* class method), 392
[has_member\(\)](#) (*elf_phdr* method), 395
[has_member\(\)](#) (*elf_phdr.VolTemplateProxy* class method), 394
[has_member\(\)](#) (*elf_sym* method), 397
[has_member\(\)](#) (*elf_sym.VolTemplateProxy* class method), 396
[has_member\(\)](#) (*Enumeration* method), 272
[has_member\(\)](#) (*Enumeration.VolTemplateProxy* class method), 270
[has_member\(\)](#) (*EPROCESS* method), 440
[has_member\(\)](#) (*EPROCESS.VolTemplateProxy* class method), 439
[has_member\(\)](#) (*ETHREAD* method), 443
[has_member\(\)](#) (*ETHREAD.VolTemplateProxy* class method), 442
[has_member\(\)](#) (*EX_FAST_REF* method), 445
[has_member\(\)](#) (*EX_FAST_REF.VolTemplateProxy* class method), 444
[has_member\(\)](#) (*ExecutiveObject* method), 491
[has_member\(\)](#) (*ExecutiveObject.VolTemplateProxy* class method), 490
[has_member\(\)](#) (*FILE_OBJECT* method), 446
[has_member\(\)](#) (*FILE_OBJECT.VolTemplateProxy* class method), 446
[has_member\(\)](#) (*fileglob* method), 406
[has_member\(\)](#) (*fileglob.VolTemplateProxy* class method), 405
[has_member\(\)](#) (*files_struct* method), 335
[has_member\(\)](#) (*files_struct.VolTemplateProxy* class method), 334
[has_member\(\)](#) (*Float* method), 275
[has_member\(\)](#) (*Float.VolTemplateProxy* class method), 274
[has_member\(\)](#) (*fs_struct* method), 336
[has_member\(\)](#) (*fs_struct.VolTemplateProxy* class method), 336
[has_member\(\)](#) (*Function* method), 277
[has_member\(\)](#) (*Function.VolTemplateProxy* class method), 276
[has_member\(\)](#) (*GenericIntelProcess* method), 320
[has_member\(\)](#) (*GenericIntelProcess.VolTemplateProxy* class method), 319
[has_member\(\)](#) (*hist_entry* method), 391
[has_member\(\)](#) (*hist_entry.VolTemplateProxy* class method), 390
[has_member\(\)](#) (*HMAP_ENTRY* method), 509
[has_member\(\)](#) (*HMAP_ENTRY.VolTemplateProxy* class method), 509
[has_member\(\)](#) (*ifnet* method), 408
[has_member\(\)](#) (*ifnet.VolTemplateProxy* class method), 407
[has_member\(\)](#) (*IMAGE_DOS_HEADER* method), 486
[has_member\(\)](#) (*IMAGE_DOS_HEADER.VolTemplateProxy* class method), 485
[has_member\(\)](#) (*IMAGE_NT_HEADERS* method), 489
[has_member\(\)](#) (*IMAGE_NT_HEADERS.VolTemplateProxy* class method), 488
[has_member\(\)](#) (*inet_sock* method), 338
[has_member\(\)](#) (*inet_sock.VolTemplateProxy* class method), 338
[has_member\(\)](#) (*inpcb* method), 410
[has_member\(\)](#) (*inpcb.VolTemplateProxy* class method), 409
[has_member\(\)](#) (*Integer* method), 280

`has_member()` (*Integer.VolTemplateProxy* class method), 278

`has_member()` (*kauth_scope* method), 411

`has_member()` (*kauth_scope.VolTemplateProxy* class method), 411

`has_member()` (*KDDEBUGGER_DATA64* method), 473

`has_member()` (*KDDEBUGGER_DATA64.VolTemplateProxy* class method), 473

`has_member()` (*kernel_cap_struct* method), 341

`has_member()` (*kernel_cap_struct.VolTemplateProxy* class method), 340

`has_member()` (*kernel_cap_t* method), 344

`has_member()` (*kernel_cap_t.VolTemplateProxy* class method), 342

`has_member()` (*KMUTANT* method), 448

`has_member()` (*KMUTANT.VolTemplateProxy* class method), 448

`has_member()` (*kobject* method), 346

`has_member()` (*kobject.VolTemplateProxy* class method), 345

`has_member()` (*KSYSTEM_TIME* method), 450

`has_member()` (*KSYSTEM_TIME.VolTemplateProxy* class method), 450

`has_member()` (*KTHREAD* method), 452

`has_member()` (*KTHREAD.VolTemplateProxy* class method), 451

`has_member()` (*LIST_ENTRY* method), 454

`has_member()` (*LIST_ENTRY.VolTemplateProxy* class method), 453

`has_member()` (*list_head* method), 347

`has_member()` (*list_head.VolTemplateProxy* class method), 347

`has_member()` (*maple_tree* method), 350

`has_member()` (*maple_tree.VolTemplateProxy* class method), 349

`has_member()` (*MFTAttribute* method), 480

`has_member()` (*MFTAttribute.VolTemplateProxy* class method), 479

`has_member()` (*MFTEntry* method), 482

`has_member()` (*MFTEntry.VolTemplateProxy* class method), 481

`has_member()` (*MFTFileName* method), 484

`has_member()` (*MFTFileName.VolTemplateProxy* class method), 483

`has_member()` (*mm_struct* method), 352

`has_member()` (*mm_struct.VolTemplateProxy* class method), 351

`has_member()` (*MMVAD* method), 457

`has_member()` (*MMVAD.VolTemplateProxy* class method), 455

`has_member()` (*MMVAD_SHORT* method), 459

`has_member()` (*MMVAD_SHORT.VolTemplateProxy* class method), 458

`has_member()` (*mnt_namespace* method), 354

`has_member()` (*mnt_namespace.VolTemplateProxy* class method), 353

`has_member()` (*module* method), 356

`has_member()` (*module.VolTemplateProxy* class method), 355

`has_member()` (*mount* method), 359

`has_member()` (*mount.VolTemplateProxy* class method), 358

`has_member()` (*net* method), 362

`has_member()` (*net.VolTemplateProxy* class method), 361

`has_member()` (*netlink_sock* method), 364

`has_member()` (*netlink_sock.VolTemplateProxy* class method), 363

`has_member()` (*OBJECT_HEADER* method), 492

`has_member()` (*OBJECT_HEADER.VolTemplateProxy* class method), 492

`has_member()` (*OBJECT_SYMBOLIC_LINK* method), 461

`has_member()` (*OBJECT_SYMBOLIC_LINK.VolTemplateProxy* class method), 460

`has_member()` (*ObjectInterface* method), 163

`has_member()` (*ObjectInterface.VolTemplateProxy* class method), 162

`has_member()` (*ObjectTemplate* method), 298

`has_member()` (*packet_sock* method), 365

`has_member()` (*packet_sock.VolTemplateProxy* class method), 365

`has_member()` (*PARTITION_ENTRY* method), 476

`has_member()` (*PARTITION_ENTRY.VolTemplateProxy* class method), 475

`has_member()` (*PARTITION_TABLE* method), 478

`has_member()` (*PARTITION_TABLE.VolTemplateProxy* class method), 477

`has_member()` (*Pointer* method), 283

`has_member()` (*Pointer.VolTemplateProxy* class method), 281

`has_member()` (*POOL_HEADER* method), 495

`has_member()` (*POOL_HEADER.VolTemplateProxy* class method), 494

`has_member()` (*POOL_HEADER_VISTA* method), 497

`has_member()` (*POOL_HEADER_VISTA.VolTemplateProxy* class method), 496

`has_member()` (*POOL_TRACKER_BIG_PAGES* method), 499

`has_member()` (*POOL_TRACKER_BIG_PAGES.VolTemplateProxy* class method), 498

`has_member()` (*PrimitiveObject* method), 285

`has_member()` (*PrimitiveObject.VolTemplateProxy* class method), 284

`has_member()` (*proc* method), 413

`has_member()` (*proc.VolTemplateProxy* class method), 412

[has_member\(\)](#) (*qstr* method), 367
[has_member\(\)](#) (*qstr.VolTemplateProxy* class method), 366
[has_member\(\)](#) (*queue_entry* method), 415
[has_member\(\)](#) (*queue_entry.VolTemplateProxy* class method), 414
[has_member\(\)](#) (*ReferenceTemplate* method), 299
[has_member\(\)](#) (*SERVICE_HEADER* method), 513
[has_member\(\)](#) (*SERVICE_HEADER.VolTemplateProxy* class method), 513
[has_member\(\)](#) (*SERVICE_RECORD* method), 516
[has_member\(\)](#) (*SERVICE_RECORD.VolTemplateProxy* class method), 515
[has_member\(\)](#) (*SHARED_CACHE_MAP* method), 463
[has_member\(\)](#) (*SHARED_CACHE_MAP.VolTemplateProxy* class method), 463
[has_member\(\)](#) (*sock* method), 369
[has_member\(\)](#) (*sock.VolTemplateProxy* class method), 368
[has_member\(\)](#) (*sockaddr* method), 417
[has_member\(\)](#) (*sockaddr.VolTemplateProxy* class method), 417
[has_member\(\)](#) (*sockaddr_dl* method), 419
[has_member\(\)](#) (*sockaddr_dl.VolTemplateProxy* class method), 418
[has_member\(\)](#) (*socket* method), 371, 421
[has_member\(\)](#) (*socket.VolTemplateProxy* class method), 370, 420
[has_member\(\)](#) (*String* method), 288
[has_member\(\)](#) (*String.VolTemplateProxy* class method), 286
[has_member\(\)](#) (*struct_file* method), 373
[has_member\(\)](#) (*struct_file.VolTemplateProxy* class method), 372
[has_member\(\)](#) (*StructType* method), 293
[has_member\(\)](#) (*StructType.VolTemplateProxy* class method), 293
[has_member\(\)](#) (*SUMMARY_DUMP* method), 472
[has_member\(\)](#) (*SUMMARY_DUMP.VolTemplateProxy* class method), 471
[has_member\(\)](#) (*super_block* method), 375
[has_member\(\)](#) (*super_block.VolTemplateProxy* class method), 374
[has_member\(\)](#) (*SymbolSpace.UnresolvedTemplate* method), 317
[has_member\(\)](#) (*sysctl_oid* method), 423
[has_member\(\)](#) (*sysctl_oid.VolTemplateProxy* class method), 422
[has_member\(\)](#) (*task_struct* method), 377
[has_member\(\)](#) (*task_struct.VolTemplateProxy* class method), 376
[has_member\(\)](#) (*Template* method), 164
[has_member\(\)](#) (*TOKEN* method), 466
[has_member\(\)](#) (*TOKEN.VolTemplateProxy* class method), 465
[has_member\(\)](#) (*UNICODE_STRING* method), 468
[has_member\(\)](#) (*UNICODE_STRING.VolTemplateProxy* class method), 467
[has_member\(\)](#) (*UnionType* method), 295
[has_member\(\)](#) (*UnionType.VolTemplateProxy* class method), 294
[has_member\(\)](#) (*unix_sock* method), 380
[has_member\(\)](#) (*unix_sock.VolTemplateProxy* class method), 379
[has_member\(\)](#) (*VACB* method), 470
[has_member\(\)](#) (*VACB.VolTemplateProxy* class method), 469
[has_member\(\)](#) (*vfsmount* method), 382
[has_member\(\)](#) (*vfsmount.VolTemplateProxy* class method), 381
[has_member\(\)](#) (*vm_area_struct* method), 385
[has_member\(\)](#) (*vm_area_struct.VolTemplateProxy* class method), 384
[has_member\(\)](#) (*vm_map_entry* method), 425
[has_member\(\)](#) (*vm_map_entry.VolTemplateProxy* class method), 424
[has_member\(\)](#) (*vm_map_object* method), 427
[has_member\(\)](#) (*vm_map_object.VolTemplateProxy* class method), 426
[has_member\(\)](#) (*vnnode* method), 428
[has_member\(\)](#) (*vnnode.VolTemplateProxy* class method), 428
[has_member\(\)](#) (*Void* method), 297
[has_member\(\)](#) (*Void.VolTemplateProxy* class method), 296
[has_member\(\)](#) (*vsock_sock* method), 387
[has_member\(\)](#) (*vsock_sock.VolTemplateProxy* class method), 386
[has_member\(\)](#) (*xdp_sock* method), 389
[has_member\(\)](#) (*xdp_sock.VolTemplateProxy* class method), 388
[has_parent\(\)](#) (*mount* method), 359
[has_parent\(\)](#) (*vfsmount* method), 382
[has_symbol\(\)](#) (*ConfigurableModule* method), 124
[has_symbol\(\)](#) (*Module* method), 128
[has_symbol\(\)](#) (*ModuleInterface* method), 152
[has_symbol\(\)](#) (*SizedModule* method), 132
[has_symbol\(\)](#) (*SymbolSpace* method), 318
[has_symbol\(\)](#) (*SymbolSpaceInterface* method), 177
[has_type\(\)](#) (*ConfigurableModule* method), 124
[has_type\(\)](#) (*Module* method), 128
[has_type\(\)](#) (*ModuleInterface* method), 152
[has_type\(\)](#) (*SizedModule* method), 132
[has_type\(\)](#) (*SymbolSpace* method), 318
[has_type\(\)](#) (*SymbolSpaceInterface* method), 177
[has_valid_member\(\)](#) (*AggregateType* method), 250
[has_valid_member\(\)](#) (*Array* method), 252
[has_valid_member\(\)](#) (*BitField* method), 255

`has_valid_member()` (*Boolean method*), 258
`has_valid_member()` (*bpf_prog method*), 327
`has_valid_member()` (*bt_sock method*), 329
`has_valid_member()` (*Bytes method*), 261
`has_valid_member()` (*Char method*), 267
`has_valid_member()` (*ClassType method*), 269
`has_valid_member()` (*CM_KEY_BODY method*), 503
`has_valid_member()` (*CM_KEY_NODE method*), 506
`has_valid_member()` (*CM_KEY_VALUE method*), 508
`has_valid_member()` (*CMHIVE method*), 501
`has_valid_member()` (*CONTROL_AREA method*), 434
`has_valid_member()` (*cred method*), 331
`has_valid_member()` (*dentry method*), 333
`has_valid_member()` (*DEVICE_OBJECT method*), 436
`has_valid_member()` (*DRIVER_OBJECT method*), 438
`has_valid_member()` (*elf method*), 393
`has_valid_member()` (*elf_phdr method*), 395
`has_valid_member()` (*elf_sym method*), 397
`has_valid_member()` (*Enumeration method*), 272
`has_valid_member()` (*EPROCESS method*), 440
`has_valid_member()` (*ETHREAD method*), 443
`has_valid_member()` (*EX_FAST_REF method*), 445
`has_valid_member()` (*ExecutiveObject method*), 491
`has_valid_member()` (*FILE_OBJECT method*), 447
`has_valid_member()` (*fileglob method*), 406
`has_valid_member()` (*files_struct method*), 335
`has_valid_member()` (*Float method*), 275
`has_valid_member()` (*fs_struct method*), 337
`has_valid_member()` (*Function method*), 277
`has_valid_member()` (*GenericIntelProcess method*), 320
`has_valid_member()` (*hist_entry method*), 391
`has_valid_member()` (*HMAP_ENTRY method*), 509
`has_valid_member()` (*ifnet method*), 408
`has_valid_member()` (*IMAGE_DOS_HEADER method*), 486
`has_valid_member()` (*IMAGE_NT_HEADERS method*), 489
`has_valid_member()` (*inet_sock method*), 339
`has_valid_member()` (*inpcb method*), 410
`has_valid_member()` (*Integer method*), 280
`has_valid_member()` (*kauth_scope method*), 411
`has_valid_member()` (*KDDEBUGGER_DATA64 method*), 474
`has_valid_member()` (*kernel_cap_struct method*), 341
`has_valid_member()` (*kernel_cap_t method*), 344
`has_valid_member()` (*KMUTANT method*), 449
`has_valid_member()` (*kobject method*), 346
`has_valid_member()` (*KSYSTEM_TIME method*), 450
`has_valid_member()` (*KTHREAD method*), 452
`has_valid_member()` (*LIST_ENTRY method*), 454
`has_valid_member()` (*list_head method*), 348
`has_valid_member()` (*maple_tree method*), 350
`has_valid_member()` (*MFTAttribute method*), 480
`has_valid_member()` (*MFTEntry method*), 482
`has_valid_member()` (*MFTFileName method*), 484
`has_valid_member()` (*mm_struct method*), 352
`has_valid_member()` (*MMVAD method*), 457
`has_valid_member()` (*MMVAD_SHORT method*), 459
`has_valid_member()` (*mnt_namespace method*), 354
`has_valid_member()` (*module method*), 356
`has_valid_member()` (*mount method*), 360
`has_valid_member()` (*net method*), 362
`has_valid_member()` (*netlink_sock method*), 364
`has_valid_member()` (*OBJECT_HEADER method*), 493
`has_valid_member()` (*OBJECT_SYMBOLIC_LINK method*), 461
`has_valid_member()` (*ObjectInterface method*), 163
`has_valid_member()` (*packet_sock method*), 365
`has_valid_member()` (*PARTITION_ENTRY method*), 476
`has_valid_member()` (*PARTITION_TABLE method*), 478
`has_valid_member()` (*Pointer method*), 283
`has_valid_member()` (*POOL_HEADER method*), 495
`has_valid_member()` (*POOL_HEADER_VISTA method*), 497
`has_valid_member()` (*POOL_TRACKER_BIG_PAGES method*), 499
`has_valid_member()` (*PrimitiveObject method*), 285
`has_valid_member()` (*proc method*), 413
`has_valid_member()` (*qstr method*), 367
`has_valid_member()` (*queue_entry method*), 415
`has_valid_member()` (*SERVICE_HEADER method*), 514
`has_valid_member()` (*SERVICE_RECORD method*), 516
`has_valid_member()` (*SHARED_CACHE_MAP method*), 464
`has_valid_member()` (*sock method*), 369
`has_valid_member()` (*sockaddr method*), 417
`has_valid_member()` (*sockaddr_dl method*), 419
`has_valid_member()` (*socket method*), 371, 421
`has_valid_member()` (*String method*), 288
`has_valid_member()` (*struct_file method*), 373
`has_valid_member()` (*StructType method*), 293
`has_valid_member()` (*SUMMARY_DUMP method*), 472
`has_valid_member()` (*super_block method*), 375
`has_valid_member()` (*sysctl_oid method*), 423
`has_valid_member()` (*task_struct method*), 377
`has_valid_member()` (*TOKEN method*), 466
`has_valid_member()` (*UNICODE_STRING method*), 468
`has_valid_member()` (*UnionType method*), 295
`has_valid_member()` (*unix_sock method*), 380

has_valid_member() (*VACB method*), 470
 has_valid_member() (*vfsmount method*), 382
 has_valid_member() (*vm_area_struct method*), 385
 has_valid_member() (*vm_map_entry method*), 425
 has_valid_member() (*vm_map_object method*), 427
 has_valid_member() (*vnode method*), 429
 has_valid_member() (*Void method*), 297
 has_valid_member() (*vsock_sock method*), 387
 has_valid_member() (*xdp_sock method*), 389
 has_valid_members() (*AggregateType method*), 250
 has_valid_members() (*Array method*), 252
 has_valid_members() (*BitField method*), 255
 has_valid_members() (*Boolean method*), 258
 has_valid_members() (*bpf_prog method*), 327
 has_valid_members() (*bt_sock method*), 329
 has_valid_members() (*Bytes method*), 261
 has_valid_members() (*Char method*), 267
 has_valid_members() (*ClassType method*), 269
 has_valid_members() (*CM_KEY_BODY method*), 504
 has_valid_members() (*CM_KEY_NODE method*), 506
 has_valid_members() (*CM_KEY_VALUE method*), 508
 has_valid_members() (*CMHIVE method*), 502
 has_valid_members() (*CONTROL_AREA method*), 434
 has_valid_members() (*cred method*), 331
 has_valid_members() (*dentry method*), 333
 has_valid_members() (*DEVICE_OBJECT method*), 436
 has_valid_members() (*DRIVER_OBJECT method*), 438
 has_valid_members() (*elf method*), 393
 has_valid_members() (*elf_phdr method*), 395
 has_valid_members() (*elf_sym method*), 397
 has_valid_members() (*Enumeration method*), 272
 has_valid_members() (*EPROCESS method*), 441
 has_valid_members() (*ETHREAD method*), 443
 has_valid_members() (*EX_FAST_REF method*), 445
 has_valid_members() (*ExecutiveObject method*), 491
 has_valid_members() (*FILE_OBJECT method*), 447
 has_valid_members() (*fileglob method*), 406
 has_valid_members() (*files_struct method*), 335
 has_valid_members() (*Float method*), 275
 has_valid_members() (*fs_struct method*), 337
 has_valid_members() (*Function method*), 277
 has_valid_members() (*GenericIntelProcess method*), 320
 has_valid_members() (*hist_entry method*), 391
 has_valid_members() (*HMAP_ENTRY method*), 510
 has_valid_members() (*ifnet method*), 408
 has_valid_members() (*IMAGE_DOS_HEADER method*), 486
 has_valid_members() (*IMAGE_NT_HEADERS method*), 489
 has_valid_members() (*inet_sock method*), 339
 has_valid_members() (*inpcb method*), 410
 has_valid_members() (*Integer method*), 280
 has_valid_members() (*kauth_scope method*), 412
 has_valid_members() (*KDDEBUGGER_DATA64 method*), 474
 has_valid_members() (*kernel_cap_struct method*), 341
 has_valid_members() (*kernel_cap_t method*), 344
 has_valid_members() (*KMUTANT method*), 449
 has_valid_members() (*kobject method*), 346
 has_valid_members() (*KSYSTEM_TIME method*), 451
 has_valid_members() (*KTHREAD method*), 453
 has_valid_members() (*LIST_ENTRY method*), 454
 has_valid_members() (*list_head method*), 348
 has_valid_members() (*maple_tree method*), 350
 has_valid_members() (*MFTAttribute method*), 480
 has_valid_members() (*MFTEntry method*), 482
 has_valid_members() (*MFTFileName method*), 484
 has_valid_members() (*mm_struct method*), 352
 has_valid_members() (*MMVAD method*), 457
 has_valid_members() (*MMVAD_SHORT method*), 459
 has_valid_members() (*mnt_namespace method*), 354
 has_valid_members() (*module method*), 356
 has_valid_members() (*mount method*), 360
 has_valid_members() (*net method*), 362
 has_valid_members() (*netlink_sock method*), 364
 has_valid_members() (*OBJECT_HEADER method*), 493
 has_valid_members() (*OBJECT_SYMBOLIC_LINK method*), 462
 has_valid_members() (*ObjectInterface method*), 163
 has_valid_members() (*packet_sock method*), 366
 has_valid_members() (*PARTITION_ENTRY method*), 477
 has_valid_members() (*PARTITION_TABLE method*), 478
 has_valid_members() (*Pointer method*), 283
 has_valid_members() (*POOL_HEADER method*), 495
 has_valid_members() (*POOL_HEADER_VISTA method*), 497
 has_valid_members() (*POOL_TRACKER_BIG_PAGES method*), 499
 has_valid_members() (*PrimitiveObject method*), 285
 has_valid_members() (*proc method*), 414
 has_valid_members() (*qstr method*), 367
 has_valid_members() (*queue_entry method*), 415
 has_valid_members() (*SERVICE_HEADER method*), 514
 has_valid_members() (*SERVICE_RECORD method*), 516

`has_valid_members()` (*SHARED_CACHE_MAP method*), 464

`has_valid_members()` (*sock method*), 369

`has_valid_members()` (*sockaddr method*), 418

`has_valid_members()` (*sockaddr_dl method*), 419

`has_valid_members()` (*socket method*), 371, 421

`has_valid_members()` (*String method*), 288

`has_valid_members()` (*struct_file method*), 373

`has_valid_members()` (*StructType method*), 294

`has_valid_members()` (*SUMMARY_DUMP method*), 472

`has_valid_members()` (*super_block method*), 375

`has_valid_members()` (*sysctl_oid method*), 423

`has_valid_members()` (*task_struct method*), 378

`has_valid_members()` (*TOKEN method*), 466

`has_valid_members()` (*UNICODE_STRING method*), 468

`has_valid_members()` (*UnionType method*), 295

`has_valid_members()` (*unix_sock method*), 380

`has_valid_members()` (*VACB method*), 470

`has_valid_members()` (*vfsmount method*), 383

`has_valid_members()` (*vm_area_struct method*), 385

`has_valid_members()` (*vm_map_entry method*), 425

`has_valid_members()` (*vm_map_object method*), 427

`has_valid_members()` (*vnnode method*), 429

`has_valid_members()` (*Void method*), 297

`has_valid_members()` (*vsock_sock method*), 387

`has_valid_members()` (*xdp_sock method*), 389

`hash` (*SizedModule property*), 132

`HASH_PTE_SIZE_64` (*QemuSuspendLayer attribute*), 230

`Hashdump` (*class in volatility3.plugins.windows.hashdump*), 686

`header_structure` (*VmwareLayer attribute*), 244

`headerpages` (*WindowsCrashDump32Layer attribute*), 186

`headerpages` (*WindowsCrashDump64Layer attribute*), 189

`help()` (*Volshell method*), 54, 57, 59, 62

`HelpfulArgParser` (*class in volatility3.cli.volargparse*), 67

`HelpfulSubparserAction` (*class in volatility3.cli.volargparse*), 68

`Hex` (*class in volatility3.framework.renderers.format_hints*), 307

`hex()` (*Bytes method*), 261

`hex()` (*Float method*), 275

`hex()` (*HexBytes method*), 309

`hex()` (*MultiTypeData method*), 313

`hex_bytes_as_text()` (*in module volatility3.cli.text_renderer*), 66

`HexBytes` (*class in volatility3.framework.renderers.format_hints*), 308

`hide_from_subclasses()` (*in module volatility3.framework*), 68

`HierarchicalDict` (*class in volatility3.framework.interfaces.configuration*), 143

`hist_entry` (*class in volatility3.framework.symbols.linux.extensions.bash*), 390

`hist_entry.VolTemplateProxy` (*class in volatility3.framework.symbols.linux.extensions.bash*), 390

`hive_offset` (*RegistryHive property*), 235

`HiveGenerator` (*class in volatility3.plugins.windows.registry.hivelist*), 647

`HiveList` (*class in volatility3.plugins.windows.registry.hivelist*), 647

`HiveScan` (*class in volatility3.plugins.windows.registry.hivescan*), 650

`HMAP_ENTRY` (*class in volatility3.framework.symbols.windows.extensions.registry*), 508

`HMAP_ENTRY.VolTemplateProxy` (*class in volatility3.framework.symbols.windows.extensions.registry*), 508

|

`IdentifierProcessor` (*class in volatility3.framework.automagic.symbol_cache*), 86

`IDENTIFIERS_FILENAME` (*in module volatility3.framework.constants*), 119

`Ifconfig` (*class in volatility3.plugins.mac.ifconfig*), 615

`ifnet` (*class in volatility3.framework.symbols.mac.extensions*), 407

`ifnet.VolTemplateProxy` (*class in volatility3.framework.symbols.mac.extensions*), 407

`imag` (*Bin attribute*), 306

`imag` (*BitField attribute*), 255

`imag` (*Boolean attribute*), 258

`imag` (*Char attribute*), 267

`imag` (*Enumeration attribute*), 273

`imag` (*Float attribute*), 275

`imag` (*Hex attribute*), 308

`imag` (*Integer attribute*), 280

`imag` (*Parallelism attribute*), 121

`imag` (*Pointer attribute*), 283

`imag` (*PoolType attribute*), 723

`imag` (*RegKeyFlags attribute*), 511

`imag` (*TimeLinerType attribute*), 763

`IMAGE_DOS_HEADER` (*class in volatility3.framework.symbols.windows.extensions.pe*),

- 485
- `IMAGE_DOS_HEADER.VolTemplateProxy` (class in `volatility3.framework.symbols.windows.extensions.pe`), 485
- `IMAGE_NT_HEADERS` (class in `volatility3.framework.symbols.windows.extensions.pe`), 487
- `IMAGE_NT_HEADERS.VolTemplateProxy` (class in `volatility3.framework.symbols.windows.extensions.pe`), 487
- `import_file()` (in module `volatility3.framework`), 68
- `import_files()` (in module `volatility3.framework`), 69
- `index()` (Array method), 252
- `index()` (Bytes method), 261
- `index()` (Column method), 169
- `index()` (DataFormatInfo method), 270
- `index()` (HexBytes method), 309
- `index()` (MountInfoData method), 592
- `index()` (MultiTypeData method), 313
- `index()` (String method), 288
- `index()` (TreeNode method), 171, 304
- `inet_ntop()` (in module `volatility3.framework.symbols.windows.extensions.networks`), 485
- `inet_sock` (class in `volatility3.framework.symbols.linux.extensions`), 337
- `inet_sock.VolTemplateProxy` (class in `volatility3.framework.symbols.linux.extensions`), 337
- `Info` (class in `volatility3.plugins.windows.info`), 689
- `init_order_modules()` (EPROCESS method), 441
- `inpcb` (class in `volatility3.framework.symbols.mac.extensions`), 408
- `inpcb.VolTemplateProxy` (class in `volatility3.framework.symbols.mac.extensions`), 409
- `instance_type` (BooleanRequirement attribute), 96
- `instance_type` (BytesRequirement attribute), 97
- `instance_type` (IntRequirement attribute), 102
- `instance_type` (SimpleTypeRequirement attribute), 147
- `instance_type` (StringRequirement attribute), 112
- `instance_type` (URIRequirement attribute), 116
- `Integer` (class in `volatility3.framework.objects`), 278
- `Integer.VolTemplateProxy` (class in `volatility3.framework.objects`), 278
- `Intel` (class in `volatility3.framework.layers.intel`), 195
- `Intel32e` (class in `volatility3.framework.layers.intel`), 197
- `IntelPAE` (class in `volatility3.framework.layers.intel`), 200
- `interface_version()` (in module `volatility3.framework`), 69
- `IntermediateSymbolTable` (class in `volatility3.framework.symbols.intermed`), 525
- `IntRequirement` (class in `volatility3.framework.configuration.requirements`), 101
- `invalid` (HiveGenerator property), 647
- `InvalidAddressException`, 555
- `invalidate_caches()` (WarningFindSpec method), 47
- `IOMem` (class in `volatility3.plugins.linux.iomem`), 575
- `is_ancestor()` (TreeGrid method), 170, 302
- `is_bootable()` (PARTITION_ENTRY method), 477
- `is_dirty()` (Intel method), 196
- `is_dirty()` (Intel32e method), 198
- `is_dirty()` (IntelPAE method), 201
- `is_dirty()` (WindowsIntel method), 204
- `is_dirty()` (WindowsIntel32e method), 206
- `is_dirty()` (WindowsIntelPAE method), 209
- `is_dirty()` (WindowsMixin method), 212
- `is_equal()` (vfsmount method), 383
- `is_free()` (POOL_TRACKER_BIG_PAGES method), 500
- `is_free_pool()` (POOL_HEADER method), 495
- `is_free_pool()` (POOL_HEADER_VISTA method), 497
- `is_integer()` (Float method), 275
- `is_kernel_thread` (task_struct property), 378
- `is_nonpaged_pool()` (POOL_HEADER method), 495
- `is_nonpaged_pool()` (POOL_HEADER_VISTA method), 497
- `is_paged_pool()` (POOL_HEADER method), 495
- `is_paged_pool()` (POOL_HEADER_VISTA method), 497
- `is_path_reachable()` (mount method), 360
- `is_readable()` (Pointer method), 283
- `is_root()` (dentry method), 333
- `is_shared()` (mount method), 360
- `is_shared()` (vfsmount method), 383
- `is_slave()` (mount method), 360
- `is_slave()` (vfsmount method), 383
- `is_subdir()` (dentry method), 333
- `is_suspicious()` (vm_area_struct method), 386
- `is_suspicious()` (vm_map_entry method), 425
- `is_thread_group_leader` (task_struct property), 378
- `is_unbindable()` (mount method), 360
- `is_unbindable()` (vfsmount method), 383
- `is_url_local()` (SqliteCache method), 88
- `is_user_thread` (task_struct property), 378
- `is_vad_empty()` (Malfind class method), 697
- `is_valid()` (AVMLLayer method), 182
- `is_valid()` (BufferDataLayer method), 226
- `is_valid()` (CMHIVE method), 502
- `is_valid()` (CONTROL_AREA method), 434

`is_valid()` (*DataLayerInterface* method), 154
`is_valid()` (*DRIVER_OBJECT* method), 438
`is_valid()` (*elf* method), 393
`is_valid()` (*Elf64Layer* method), 192
`is_valid()` (*EPROCESS* method), 441
`is_valid()` (*FILE_OBJECT* method), 447
`is_valid()` (*FileLayer* method), 228
`is_valid()` (*hist_entry* method), 391
`is_valid()` (*Intel* method), 196
`is_valid()` (*Intel32e* method), 198
`is_valid()` (*IntelPAE* method), 201
`is_valid()` (*KMUTANT* method), 449
`is_valid()` (*LimeLayer* method), 215
`is_valid()` (*LinearlyMappedLayer* method), 218
`is_valid()` (*NonLinearlySegmentedLayer* method), 239
`is_valid()` (*OBJECT_HEADER* method), 493
`is_valid()` (*OBJECT_SYMBOLIC_LINK* method), 462
`is_valid()` (*PdbMSFStream* method), 220
`is_valid()` (*PdbMultiStreamFormat* method), 223
`is_valid()` (*POOL_TRACKER_BIG_PAGES* method), 500
`is_valid()` (*QemuSuspendLayer* method), 231
`is_valid()` (*RegistryHive* method), 235
`is_valid()` (*SegmentedLayer* method), 242
`is_valid()` (*SERVICE_HEADER* method), 514
`is_valid()` (*SERVICE_RECORD* method), 516
`is_valid()` (*SHARED_CACHE_MAP* method), 464
`is_valid()` (*TranslationLayerInterface* method), 159
`is_valid()` (*vfsmount* method), 383
`is_valid()` (*VmwareLayer* method), 244
`is_valid()` (*WindowsCrashDump32Layer* method), 186
`is_valid()` (*WindowsCrashDump64Layer* method), 189
`is_valid()` (*WindowsIntel* method), 204
`is_valid()` (*WindowsIntel32e* method), 207
`is_valid()` (*WindowsIntelPAE* method), 209
`is_valid()` (*WindowsMixin* method), 212
`is_valid()` (*XenCoreDumpLayer* method), 247
`is_valid_choice` (*Enumeration* property), 273
`isalnum()` (*Bytes* method), 262
`isalnum()` (*HexBytes* method), 309
`isalnum()` (*MultiTypeData* method), 313
`isalnum()` (*String* method), 288
`isalpha()` (*Bytes* method), 262
`isalpha()` (*HexBytes* method), 309
`isalpha()` (*MultiTypeData* method), 313
`isalpha()` (*String* method), 288
`isascii()` (*Bytes* method), 262
`isascii()` (*HexBytes* method), 309
`isascii()` (*MultiTypeData* method), 313
`isascii()` (*String* method), 289
`isatty()` (*FileHandlerInterface* method), 165
`isatty()` (*NullFileHandler* method), 51
`isdecimal()` (*String* method), 289
`isdigit()` (*Bytes* method), 262
`isdigit()` (*HexBytes* method), 309
`isdigit()` (*MultiTypeData* method), 313
`isdigit()` (*String* method), 289
`ISF_EXTENSIONS` (in module *volatility3.framework.constants*), 119
`ISF_MINIMUM_DEPRECATED` (in module *volatility3.framework.constants*), 119
`ISF_MINIMUM_SUPPORTED` (in module *volatility3.framework.constants*), 120
`IsfInfo` (class in *volatility3.plugins.isfinfo*), 758
`ISFormatTable` (class in *volatility3.framework.symbols.intermed*), 523
`isidentifier()` (*String* method), 289
`islower()` (*Bytes* method), 262
`islower()` (*HexBytes* method), 309
`islower()` (*MultiTypeData* method), 313
`islower()` (*String* method), 289
`isnumeric()` (*String* method), 289
`isprintable()` (*String* method), 289
`isspace()` (*Bytes* method), 262
`isspace()` (*HexBytes* method), 309
`isspace()` (*MultiTypeData* method), 313
`isspace()` (*String* method), 289
`istitle()` (*Bytes* method), 262
`istitle()` (*HexBytes* method), 309
`istitle()` (*MultiTypeData* method), 313
`istitle()` (*String* method), 289
`isupper()` (*Bytes* method), 262
`isupper()` (*HexBytes* method), 310
`isupper()` (*MultiTypeData* method), 314
`isupper()` (*String* method), 289
`items()` (*HierarchicalDict* method), 144
`items()` (*LayerContainer* method), 157
`items()` (*ModuleCollection* method), 131
`items()` (*ModuleContainer* method), 150
`items()` (*ObjectInformation* method), 161
`items()` (*ReadOnlyMapping* method), 163
`items()` (*SymbolSpace* method), 318
`items()` (*SymbolSpaceInterface* method), 177

J

`JarHandler` (class in *volatility3.framework.layers.resources*), 237
`JobLinks` (class in *volatility3.plugins.windows.joblinks*), 691
`join()` (*Bytes* method), 262
`join()` (*HexBytes* method), 310
`join()` (*MultiTypeData* method), 314
`join()` (*String* method), 289
`JsonLinesRenderer` (class in *volatility3.cli.text_renderer*), 64
`JsonRenderer` (class in *volatility3.cli.text_renderer*), 65

K

- `Kauth_listeners` (class in `volatility3.plugins.mac.kauth_listeners`), 616
- `kauth_scope` (class in `volatility3.framework.symbols.mac.extensions`), 410
- `kauth_scope.VolTemplateProxy` (class in `volatility3.framework.symbols.mac.extensions`), 410
- `Kauth_scopes` (class in `volatility3.plugins.mac.kauth_scopes`), 618
- `KDDEBUGGER_DATA64` (class in `volatility3.framework.symbols.windows.extensions.kdbg`), 472
- `KDDEBUGGER_DATA64.VolTemplateProxy` (class in `volatility3.framework.symbols.windows.extensions.kdbg`), 472
- `kernel` (Volshell property), 54, 57, 59, 62
- `kernel_cap_struct` (class in `volatility3.framework.symbols.linux.extensions`), 339
- `kernel_cap_struct.VolTemplateProxy` (class in `volatility3.framework.symbols.linux.extensions`), 339
- `kernel_cap_t` (class in `volatility3.framework.symbols.linux.extensions`), 342
- `kernel_cap_t.VolTemplateProxy` (class in `volatility3.framework.symbols.linux.extensions`), 342
- `KERNEL_MODULE_NAMES` (in module `volatility3.framework.constants.windows`), 122
- `KernelModule` (class in `volatility3.framework.automagic.module`), 77
- `KernelPDBScanner` (class in `volatility3.framework.automagic.pdbscan`), 78
- `Kevents` (class in `volatility3.plugins.mac.kevents`), 620
- `KEY_COMP_NAME` (RegKeyFlags attribute), 510
- `KEY_HIVE_ENTRY` (RegKeyFlags attribute), 510
- `KEY_HIVE_EXIT` (RegKeyFlags attribute), 510
- `KEY_IS_VOLATILE` (RegKeyFlags attribute), 510
- `key_iterator()` (PrintKey class method), 652
- `KEY_NO_DELETE` (RegKeyFlags attribute), 510
- `KEY_PREFEF_HANDLE` (RegKeyFlags attribute), 510
- `KEY_SYM_LINK` (RegKeyFlags attribute), 510
- `KEY_VIRT_MIRRORED` (RegKeyFlags attribute), 510
- `KEY_VIRT_TARGET` (RegKeyFlags attribute), 510
- `KEY_VIRTUAL_STORE` (RegKeyFlags attribute), 510
- `Keyboard_notifiers` (class in `volatility3.plugins.linux.keyboard_notifiers`), 577
- `keys()` (HierarchicalDict method), 144
- `keys()` (LayerContainer method), 157
- `keys()` (ModuleCollection method), 131
- `keys()` (ModuleContainer method), 150
- `keys()` (ObjectInformation method), 161
- `keys()` (ReadOnlyMapping method), 163
- `keys()` (SymbolSpace method), 319
- `keys()` (SymbolSpaceInterface method), 177
- `Kmsg` (class in `volatility3.plugins.linux.kmsg`), 580
- `KmsgFiveTen` (class in `volatility3.plugins.linux.kmsg`), 581
- `KmsgLegacy` (class in `volatility3.plugins.linux.kmsg`), 583
- `KMUTANT` (class in `volatility3.framework.symbols.windows.extensions`), 447
- `KMUTANT.VolTemplateProxy` (class in `volatility3.framework.symbols.windows.extensions`), 447
- `kobject` (class in `volatility3.framework.symbols.linux.extensions`), 345
- `kobject.VolTemplateProxy` (class in `volatility3.framework.symbols.linux.extensions`), 345
- `KSYSTEM_TIME` (class in `volatility3.framework.symbols.windows.extensions`), 449
- `KSYSTEM_TIME.VolTemplateProxy` (class in `volatility3.framework.symbols.windows.extensions`), 449
- `KTHREAD` (class in `volatility3.framework.symbols.windows.extensions`), 451
- `KTHREAD.VolTemplateProxy` (class in `volatility3.framework.symbols.windows.extensions`), 451

L

- `layer_name` (BytesScanner property), 180
- `layer_name` (ConfigurableModule property), 124
- `layer_name` (Module property), 128
- `layer_name` (ModuleInterface property), 152
- `layer_name` (MultiStringScanner property), 180
- `layer_name` (PageMapScanner property), 93
- `layer_name` (PdbSignatureScanner property), 522
- `layer_name` (PoolHeaderScanner property), 719
- `layer_name` (RegExScanner property), 181
- `layer_name` (ScannerInterface property), 158
- `layer_name` (SizedModule property), 132
- `layer_name` (YaraScanner property), 767
- `LayerContainer` (class in `volatility3.framework.interfaces.layers`), 156
- `LayerException`, 556
- `LayerListRequirement` (class in `volatility3.framework.configuration.requirements`), 102
- `layers` (Context property), 126
- `layers` (ContextInterface property), 149

LayerStacker (class in volatility3.framework.automagic.stacker), 82

LayerWriter (class in volatility3.plugins.layerwriter), 760

LdrModules (class in volatility3.plugins.windows.ldrmodules), 693

length (DataFormatInfo attribute), 270

LEVELS (ABCKmsg attribute), 578

LEVELS (KmsgFiveTen attribute), 582

LEVELS (KmsgLegacy attribute), 583

LimeFormatException, 214

LimeLayer (class in volatility3.framework.layers.lime), 214

LimeStacker (class in volatility3.framework.layers.lime), 216

LinearlyMappedLayer (class in volatility3.framework.layers.linear), 217

LinuxIdentifier (class in volatility3.framework.automagic.symbol_cache), 86

LinuxIntelStacker (class in volatility3.framework.automagic.linux), 72

LinuxKernelIntermedSymbols (class in volatility3.framework.symbols.linux), 321

LinuxMetadata (class in volatility3.framework.symbols.metadata), 553

LinuxSymbolFinder (class in volatility3.framework.automagic.linux), 72

LinuxUtilities (class in volatility3.framework.symbols.linux), 324

list_all_isf_files() (IsfInfo class method), 759

list_big_pools() (BigPools class method), 656

list_bugcheck_callbacks() (Callbacks class method), 660

list_bugcheck_reason_callbacks() (Callbacks class method), 660

LIST_ENTRY (class in volatility3.framework.symbols.windows.extensions), 453

LIST_ENTRY.VolTemplateProxy (class in volatility3.framework.symbols.windows.extensions), 453

list_fds() (Lsof class method), 587

List_Files (class in volatility3.plugins.mac.list_files), 622

list_files() (List_Files class method), 622

list_handlers (ResourceAccessor attribute), 238

list_head (class in volatility3.framework.symbols.linux.extensions), 346

list_head.VolTemplateProxy (class in volatility3.framework.symbols.linux.extensions), 347

list_hive_objects() (HiveList class method), 648

list_hives() (HiveList class method), 648

list_injections() (Malfind class method), 697

list_kauth_scopes() (Kauth_scopes class method), 618

list_kernel_events() (Kevents class method), 620

list_modules() (Lsmmod class method), 585, 624

list_modules() (Modules class method), 708

list_mounts() (Mount class method), 629

list_notify_routines() (Callbacks class method), 661

list_plugins() (in module volatility3.framework), 69

list_processes() (PsList class method), 727

list_processes() (Volshell method), 62

list_registry_callbacks() (Callbacks class method), 661

list_sockets() (NetStat class method), 716

list_sockets() (Netstat class method), 630

list_sockets() (Sockstat class method), 603

list_tasks() (PsList class method), 597

list_tasks() (Volshell method), 57, 60

list_tasks_allproc() (PsList class method), 636

list_tasks_pid_hash_table() (PsList class method), 636

list_tasks_process_group() (PsList class method), 636

list_tasks_sessions() (PsList class method), 637

list_tasks_tasks() (PsList class method), 637

list_userassist() (UserAssist method), 654

list_vads() (VadInfo class method), 744

list_vmas() (Maps class method), 593

ListRequirement (class in volatility3.framework.configuration.requirements), 104

ljust() (Bytes method), 262

ljust() (HexBytes method), 310

ljust() (MultiTypeData method), 314

ljust() (String method), 290

lmkey (Hashdump attribute), 687

load_cached_validations() (in module volatility3.schemas), 767

load_file() (Volshell method), 54, 57, 60, 63

load_order_modules() (EPROCESS method), 441

load_pdb_layer() (PdbReader class method), 518

load_windows_symbol_table() (PDBUtility class method), 520

locate_banners() (Banners class method), 754

location (FileLayer property), 228

location_from_file() (CommandLine class method), 48

location_from_file() (URIRequirement class method), 116

location_from_file() (VolShell class method), 49

LOGLEVEL_V (in module volatility3.framework.constants), 120

- LOGLEVEL_VV (in module *volatility3.framework.constants*), 120
- LOGLEVEL_VVV (in module *volatility3.framework.constants*), 120
- LOGLEVEL_VVVV (in module *volatility3.framework.constants*), 120
- lookup() (Enumeration method), 273
- lookup() (Enumeration.VolTemplateProxy class method), 270
- lookup_module_address() (LinuxUtilities class method), 325
- lookup_module_address() (MacUtilities class method), 404
- lookup_user_sids() (GetSIDs method), 682
- lower() (Bytes method), 262
- lower() (HexBytes method), 310
- lower() (MultiTypeData method), 314
- lower() (String method), 290
- Lsadump (class in *volatility3.plugins.windows.lsadump*), 694
- Lsmode (class in *volatility3.plugins.linux.lsmode*), 585
- Lsmode (class in *volatility3.plugins.mac.lsmode*), 623
- Lsof (class in *volatility3.plugins.linux.lsof*), 587
- Lsof (class in *volatility3.plugins.mac.lsof*), 625
- lstrip() (Bytes method), 262
- lstrip() (HexBytes method), 310
- lstrip() (MultiTypeData method), 314
- lstrip() (String method), 290
- ## M
- MacIdentifier (class in *volatility3.framework.automagic.symbol_cache*), 86
- MacIntelStacker (class in *volatility3.framework.automagic.mac*), 74
- MacKernelIntermedSymbols (class in *volatility3.framework.symbols.mac*), 401
- MacSymbolFinder (class in *volatility3.framework.automagic.mac*), 75
- MacUtilities (class in *volatility3.framework.symbols.mac*), 404
- MAGIC (Elf64Layer attribute), 191
- MAGIC (LimeLayer attribute), 214
- MAGIC (XenCoreDumpLayer attribute), 246
- main() (in module *volatility3.cli*), 49
- main() (in module *volatility3.cli.volshell*), 50
- major (super_block property), 376
- make_subconfig() (ADS class method), 702
- make_subconfig() (AutomagicInterface class method), 135
- make_subconfig() (AVMLLayer class method), 183
- make_subconfig() (Banners class method), 754
- make_subconfig() (Bash class method), 559, 609
- make_subconfig() (BashIntermedSymbols class method), 399
- make_subconfig() (BigPools class method), 656
- make_subconfig() (BufferDataLayer class method), 226
- make_subconfig() (Cachedump class method), 658
- make_subconfig() (Callbacks class method), 661
- make_subconfig() (Capabilities class method), 561
- make_subconfig() (Check_afinfo class method), 564
- make_subconfig() (Check_creds class method), 565
- make_subconfig() (Check_idt class method), 567
- make_subconfig() (Check_modules class method), 568
- make_subconfig() (Check_syscall class method), 570, 610
- make_subconfig() (Check_sysctl class method), 612
- make_subconfig() (Check_trap_table class method), 614
- make_subconfig() (CmdLine class method), 663
- make_subconfig() (ConfigurableInterface class method), 139
- make_subconfig() (ConfigurableModule class method), 124
- make_subconfig() (ConfigWriter class method), 756
- make_subconfig() (ConstructionMagic class method), 71
- make_subconfig() (Crashinfo class method), 664
- make_subconfig() (DataLayerInterface class method), 155
- make_subconfig() (DeviceTree class method), 666
- make_subconfig() (DllList class method), 668
- make_subconfig() (DriverIrp class method), 670
- make_subconfig() (DriverModule class method), 671
- make_subconfig() (DriverScan class method), 673
- make_subconfig() (DumpFiles class method), 675
- make_subconfig() (Elf64Layer class method), 192
- make_subconfig() (Elfs class method), 572
- make_subconfig() (Envars class method), 573, 677
- make_subconfig() (FileLayer class method), 228
- make_subconfig() (FileScan class method), 679
- make_subconfig() (FrameworkInfo class method), 757
- make_subconfig() (GetServiceSIDs class method), 681
- make_subconfig() (GetSIDs class method), 683
- make_subconfig() (Handles class method), 685
- make_subconfig() (Hashdump class method), 687
- make_subconfig() (HiveList class method), 649
- make_subconfig() (HiveScan class method), 650
- make_subconfig() (Ifconfig class method), 615
- make_subconfig() (Info class method), 690
- make_subconfig() (Intel class method), 196
- make_subconfig() (Intel32e class method), 199
- make_subconfig() (IntelPAE class method), 201
- make_subconfig() (IntermediateSymbolTable class method), 528
- make_subconfig() (IOMem class method), 575

`make_subconfig()` (*IsfInfo* class method), 759
`make_subconfig()` (*ISFormatTable* class method), 524
`make_subconfig()` (*JobLinks* class method), 692
`make_subconfig()` (*Kauth_listeners* class method), 617
`make_subconfig()` (*Kauth_scopes* class method), 618
`make_subconfig()` (*KernelModule* class method), 77
`make_subconfig()` (*KernelPDBScanner* class method), 80
`make_subconfig()` (*Kevents* class method), 621
`make_subconfig()` (*Keyboard_notifiers* class method), 577
`make_subconfig()` (*Kmsg* class method), 580
`make_subconfig()` (*LayerStacker* class method), 83
`make_subconfig()` (*LayerWriter* class method), 760
`make_subconfig()` (*LdrModules* class method), 693
`make_subconfig()` (*LimeLayer* class method), 215
`make_subconfig()` (*LinearlyMappedLayer* class method), 218
`make_subconfig()` (*LinuxKernelIntermedSymbols* class method), 323
`make_subconfig()` (*LinuxSymbolFinder* class method), 73
`make_subconfig()` (*List_Files* class method), 622
`make_subconfig()` (*Lsadump* class method), 695
`make_subconfig()` (*Lsmodule* class method), 586, 624
`make_subconfig()` (*Lsof* class method), 587, 626
`make_subconfig()` (*MacKernelIntermedSymbols* class method), 403
`make_subconfig()` (*MacSymbolFinder* class method), 76
`make_subconfig()` (*Malfind* class method), 589, 627, 697
`make_subconfig()` (*Maps* class method), 593, 632
`make_subconfig()` (*MBRScan* class method), 699
`make_subconfig()` (*Memmap* class method), 701
`make_subconfig()` (*MFTScan* class method), 704
`make_subconfig()` (*ModScan* class method), 706
`make_subconfig()` (*Module* class method), 128
`make_subconfig()` (*ModuleInterface* class method), 152
`make_subconfig()` (*Modules* class method), 709
`make_subconfig()` (*Mount* class method), 629
`make_subconfig()` (*MountInfo* class method), 590
`make_subconfig()` (*MutantScan* class method), 710
`make_subconfig()` (*NetScan* class method), 713
`make_subconfig()` (*NetStat* class method), 717
`make_subconfig()` (*Netstat* class method), 631
`make_subconfig()` (*NonLinearlySegmentedLayer* class method), 239
`make_subconfig()` (*PdbMSFStream* class method), 221
`make_subconfig()` (*PdbMultiStreamFormat* class method), 223
`make_subconfig()` (*PluginInterface* class method), 167
`make_subconfig()` (*PoolScanner* class method), 721
`make_subconfig()` (*PrintKey* class method), 652
`make_subconfig()` (*Privs* class method), 725
`make_subconfig()` (*PsAux* class method), 595
`make_subconfig()` (*Psaux* class method), 634
`make_subconfig()` (*PsList* class method), 598, 637, 727
`make_subconfig()` (*PsScan* class method), 599, 729
`make_subconfig()` (*PsTree* class method), 601, 639, 732
`make_subconfig()` (*QemuSuspendLayer* class method), 232
`make_subconfig()` (*RegistryHive* class method), 235
`make_subconfig()` (*SegmentedLayer* class method), 242
`make_subconfig()` (*Sessions* class method), 733
`make_subconfig()` (*SizedModule* class method), 132
`make_subconfig()` (*Skeleton_Key_Check* class method), 735
`make_subconfig()` (*Socket_filters* class method), 641
`make_subconfig()` (*Sockstat* class method), 603
`make_subconfig()` (*SSDT* class method), 737
`make_subconfig()` (*Strings* class method), 739
`make_subconfig()` (*SvcScan* class method), 741
`make_subconfig()` (*SymbolCacheMagic* class method), 89
`make_subconfig()` (*SymbolFinder* class method), 91
`make_subconfig()` (*SymbolTableInterface* class method), 179
`make_subconfig()` (*SymlinkScan* class method), 742
`make_subconfig()` (*Timeliner* class method), 764
`make_subconfig()` (*Timers* class method), 642
`make_subconfig()` (*TranslationLayerInterface* class method), 159
`make_subconfig()` (*Trustedbsd* class method), 644
`make_subconfig()` (*tty_check* class method), 605
`make_subconfig()` (*UserAssist* class method), 654
`make_subconfig()` (*VadInfo* class method), 744
`make_subconfig()` (*VadWalk* class method), 747
`make_subconfig()` (*VadYaraScan* class method), 748
`make_subconfig()` (*VerInfo* class method), 750
`make_subconfig()` (*Version1Format* class method), 531
`make_subconfig()` (*Version2Format* class method), 534
`make_subconfig()` (*Version3Format* class method), 537
`make_subconfig()` (*Version4Format* class method), 540
`make_subconfig()` (*Version5Format* class method), 543
`make_subconfig()` (*Version6Format* class method), 546
`make_subconfig()` (*Version7Format* class method), 549
`make_subconfig()` (*Version8Format* class method), 552
`make_subconfig()` (*VFSevents* class method), 646
`make_subconfig()` (*VirtMap* class method), 752
`make_subconfig()` (*VmaYaraScan* class method), 607
`make_subconfig()` (*VmwareLayer* class method), 244
`make_subconfig()` (*Volshell* class method), 54, 57, 60, 63

- `make_subconfig()` (*WindowsCrashDump32Layer* class method), 186
- `make_subconfig()` (*WindowsCrashDump64Layer* class method), 189
- `make_subconfig()` (*WindowsIntel* class method), 204
- `make_subconfig()` (*WindowsIntel32e* class method), 207
- `make_subconfig()` (*WindowsIntelPAE* class method), 209
- `make_subconfig()` (*WindowsKernelIntermedSymbols* class method), 431
- `make_subconfig()` (*WindowsMixin* class method), 212
- `make_subconfig()` (*WinSwapLayers* class method), 94
- `make_subconfig()` (*XenCoreDumpLayer* class method), 247
- `make_subconfig()` (*YaraScan* class method), 766
- `maketrans()` (*Bytes* static method), 262
- `maketrans()` (*HexBytes* static method), 310
- `maketrans()` (*MultiTypeData* static method), 314
- `maketrans()` (*String* static method), 290
- Malfind* (class in *volatility3.plugins.linux.malfind*), 588
- Malfind* (class in *volatility3.plugins.mac.malfind*), 627
- Malfind* (class in *volatility3.plugins.windows.malfind*), 696
- MAPLE_ARANGE_64* (*maple_tree* attribute), 349
- MAPLE_DENSE* (*maple_tree* attribute), 349
- MAPLE_LEAF_64* (*maple_tree* attribute), 349
- MAPLE_NODE_POINTER_MASK* (*maple_tree* attribute), 349
- MAPLE_NODE_TYPE_MASK* (*maple_tree* attribute), 349
- MAPLE_NODE_TYPE_SHIFT* (*maple_tree* attribute), 349
- MAPLE_RANGE_64* (*maple_tree* attribute), 349
- maple_tree* (class in *volatility3.framework.symbols.linux.extensions*), 348
- maple_tree.VolTemplateProxy* (class in *volatility3.framework.symbols.linux.extensions*), 349
- `mapping()` (*AVMLLayer* method), 183
- `mapping()` (*Elf64Layer* method), 192
- `mapping()` (*Intel* method), 196
- `mapping()` (*Intel32e* method), 199
- `mapping()` (*IntelPAE* method), 201
- `mapping()` (*LimeLayer* method), 215
- `mapping()` (*LinearlyMappedLayer* method), 218
- `mapping()` (*NonLinearlySegmentedLayer* method), 240
- `mapping()` (*PdbMSFStream* method), 221
- `mapping()` (*PdbMultiStreamFormat* method), 224
- `mapping()` (*QemuSuspendLayer* method), 232
- `mapping()` (*RegistryHive* method), 235
- `mapping()` (*SegmentedLayer* method), 242
- `mapping()` (*TranslationLayerInterface* method), 159
- `mapping()` (*VmwareLayer* method), 245
- `mapping()` (*WindowsCrashDump32Layer* method), 186
- `mapping()` (*WindowsCrashDump64Layer* method), 189
- `mapping()` (*WindowsIntel* method), 204
- `mapping()` (*WindowsIntel32e* method), 207
- `mapping()` (*WindowsIntelPAE* method), 210
- `mapping()` (*WindowsMixin* method), 212
- `mapping()` (*XenCoreDumpLayer* method), 247
- Maps* (class in *volatility3.plugins.linux.proc*), 592
- Maps* (class in *volatility3.plugins.mac.proc_maps*), 632
- `mask_mods_list()` (*LinuxUtilities* class method), 325
- `mask_mods_list()` (*MacUtilities* class method), 404
- `matches_required()` (*PluginRequirement* class method), 110
- `matches_required()` (*VersionRequirement* class method), 118
- `max_depth()` (*TreeGrid* method), 170, 302
- `max_pdb_size` (*KernelPDBScanner* attribute), 80
- `maximum_address` (*AVMLLayer* property), 183
- `maximum_address` (*BufferDataLayer* property), 226
- `maximum_address` (*DataLayerInterface* property), 155
- `maximum_address` (*Elf64Layer* property), 193
- `maximum_address` (*FileLayer* property), 229
- `maximum_address` (*Intel* attribute), 196
- `maximum_address` (*Intel32e* attribute), 199
- `maximum_address` (*IntelPAE* attribute), 202
- `maximum_address` (*LimeLayer* property), 215
- `maximum_address` (*LinearlyMappedLayer* property), 218
- `maximum_address` (*NonLinearlySegmentedLayer* property), 240
- `maximum_address` (*PdbMSFStream* property), 221
- `maximum_address` (*PdbMultiStreamFormat* property), 224
- `maximum_address` (*QemuSuspendLayer* property), 232
- `maximum_address` (*RegistryHive* property), 235
- `maximum_address` (*SegmentedLayer* property), 242
- `maximum_address` (*TranslationLayerInterface* property), 160
- `maximum_address` (*VmwareLayer* property), 245
- `maximum_address` (*WindowsCrashDump32Layer* property), 186
- `maximum_address` (*WindowsCrashDump64Layer* property), 189
- `maximum_address` (*WindowsIntel* attribute), 204
- `maximum_address` (*WindowsIntel32e* attribute), 207
- `maximum_address` (*WindowsIntelPAE* attribute), 210
- `maximum_address` (*WindowsMixin* attribute), 213
- `maximum_address` (*XenCoreDumpLayer* property), 247
- MAXSIZE_DEFAULT* (*Maps* attribute), 592
- MAXSIZE_DEFAULT* (*VadInfo* attribute), 744
- MBRScan* (class in *volatility3.plugins.windows.mbrscan*), 698
- `mem_order_modules()` (*EPROCESS* method), 441
- `member()` (*AggregateType* method), 251
- `member()` (*bpf_prog* method), 327
- `member()` (*bt_sock* method), 329

`member()` (*ClassType* method), 269
`member()` (*CM_KEY_BODY* method), 504
`member()` (*CM_KEY_NODE* method), 506
`member()` (*CM_KEY_VALUE* method), 508
`member()` (*CMHIVE* method), 502
`member()` (*CONTROL_AREA* method), 434
`member()` (*cred* method), 331
`member()` (*dentry* method), 333
`member()` (*DEVICE_OBJECT* method), 436
`member()` (*DRIVER_OBJECT* method), 438
`member()` (*elf* method), 393
`member()` (*elf_phdr* method), 395
`member()` (*elf_sym* method), 397
`member()` (*EPROCESS* method), 441
`member()` (*ETHREAD* method), 443
`member()` (*EX_FAST_REF* method), 445
`member()` (*FILE_OBJECT* method), 447
`member()` (*fileglob* method), 406
`member()` (*files_struct* method), 335
`member()` (*fs_struct* method), 337
`member()` (*GenericIntelProcess* method), 321
`member()` (*hist_entry* method), 391
`member()` (*HMAP_ENTRY* method), 510
`member()` (*ifnet* method), 408
`member()` (*IMAGE_DOS_HEADER* method), 487
`member()` (*IMAGE_NT_HEADERS* method), 489
`member()` (*inet_sock* method), 339
`member()` (*inpcb* method), 410
`member()` (*kauth_scope* method), 412
`member()` (*KDDEBUGGER_DATA64* method), 474
`member()` (*kernel_cap_struct* method), 342
`member()` (*kernel_cap_t* method), 344
`member()` (*KMUTANT* method), 449
`member()` (*kobject* method), 346
`member()` (*KSYSTEM_TIME* method), 451
`member()` (*KTHREAD* method), 453
`member()` (*LIST_ENTRY* method), 454
`member()` (*list_head* method), 348
`member()` (*maple_tree* method), 350
`member()` (*MFTAttribute* method), 481
`member()` (*MFTEntry* method), 482
`member()` (*MFTFileName* method), 484
`member()` (*mm_struct* method), 352
`member()` (*MMVAD* method), 457
`member()` (*MMVAD_SHORT* method), 460
`member()` (*mnt_namespace* method), 354
`member()` (*module* method), 356
`member()` (*mount* method), 360
`member()` (*net* method), 362
`member()` (*netlink_sock* method), 364
`member()` (*OBJECT_HEADER* method), 493
`member()` (*OBJECT_SYMBOLIC_LINK* method), 462
`member()` (*packet_sock* method), 366
`member()` (*PARTITION_ENTRY* method), 477
`member()` (*PARTITION_TABLE* method), 479
`member()` (*POOL_HEADER* method), 495
`member()` (*POOL_HEADER_VISTA* method), 498
`member()` (*POOL_TRACKER_BIG_PAGES* method), 500
`member()` (*proc* method), 414
`member()` (*qstr* method), 367
`member()` (*queue_entry* method), 415
`member()` (*SERVICE_HEADER* method), 514
`member()` (*SERVICE_RECORD* method), 516
`member()` (*SHARED_CACHE_MAP* method), 464
`member()` (*sock* method), 369
`member()` (*sockaddr* method), 418
`member()` (*sockaddr_dl* method), 419
`member()` (*socket* method), 371, 421
`member()` (*struct_file* method), 373
`member()` (*StructType* method), 294
`member()` (*SUMMARY_DUMP* method), 472
`member()` (*super_block* method), 376
`member()` (*sysctl_oid* method), 423
`member()` (*task_struct* method), 378
`member()` (*TOKEN* method), 466
`member()` (*UNICODE_STRING* method), 468
`member()` (*UnionType* method), 296
`member()` (*unix_sock* method), 380
`member()` (*VACB* method), 470
`member()` (*vfsmount* method), 383
`member()` (*vm_area_struct* method), 386
`member()` (*vm_map_entry* method), 425
`member()` (*vm_map_object* method), 427
`member()` (*vnode* method), 429
`member()` (*vsock_sock* method), 387
`member()` (*xdp_sock* method), 389
`Memmap` (*class in volatility3.plugins.windows.memmap*), 700
`merge()` (*HierarchicalDict* method), 144
`metadata` (*AVMLLayer* property), 183
`metadata` (*BashIntermedSymbols* property), 400
`metadata` (*BufferDataLayer* property), 226
`metadata` (*DataLayerInterface* property), 155
`metadata` (*Elf64Layer* property), 193
`metadata` (*FileLayer* property), 229
`metadata` (*Intel* property), 196
`metadata` (*Intel32e* property), 199
`metadata` (*IntelPAE* property), 202
`metadata` (*IntermediateSymbolTable* property), 528
`metadata` (*ISFormatTable* property), 525
`metadata` (*LimeLayer* property), 215
`metadata` (*LinearlyMappedLayer* property), 218
`metadata` (*LinuxKernelIntermedSymbols* property), 323
`metadata` (*MacKernelIntermedSymbols* property), 403
`metadata` (*NonLinearlySegmentedLayer* property), 240
`metadata` (*PdbMSFStream* property), 221
`metadata` (*PdbMultiStreamFormat* property), 224

- metadata (*QemuSuspendLayer* property), 232
- metadata (*RegistryHive* property), 235
- metadata (*SegmentedLayer* property), 242
- metadata (*TranslationLayerInterface* property), 160
- metadata (*Version1Format* property), 531
- metadata (*Version2Format* property), 534
- metadata (*Version3Format* property), 537
- metadata (*Version4Format* property), 540
- metadata (*Version5Format* property), 543
- metadata (*Version6Format* property), 546
- metadata (*Version7Format* property), 549
- metadata (*Version8Format* property), 552
- metadata (*VmwareLayer* property), 245
- metadata (*WindowsCrashDump32Layer* property), 186
- metadata (*WindowsCrashDump64Layer* property), 189
- metadata (*WindowsIntel* property), 204
- metadata (*WindowsIntel32e* property), 207
- metadata (*WindowsIntelPAE* property), 210
- metadata (*WindowsKernelIntermedSymbols* property), 432
- metadata (*WindowsMixin* property), 213
- metadata (*XenCoreDumpLayer* property), 247
- MetadataInterface (class in *volatility3.framework.interfaces.symbols*), 173
- method_fixed_mapping() (*KernelPDBScanner* method), 81
- method_kdbg_offset() (*KernelPDBScanner* method), 81
- method_module_offset() (*KernelPDBScanner* method), 81
- method_slow_scan() (*KernelPDBScanner* method), 81
- methods (*KernelPDBScanner* attribute), 81
- MFTAttribute (class in *volatility3.framework.symbols.windows.extensions.mft*), 479
- MFTAttribute.VolTemplateProxy (class in *volatility3.framework.symbols.windows.extensions.mft*), 479
- MFTEntry (class in *volatility3.framework.symbols.windows.extensions.mft*), 481
- MFTEntry.VolTemplateProxy (class in *volatility3.framework.symbols.windows.extensions.mft*), 481
- MFTFileName (class in *volatility3.framework.symbols.windows.extensions.mft*), 483
- MFTFileName.VolTemplateProxy (class in *volatility3.framework.symbols.windows.extensions.mft*), 483
- MFTScan (class in *volatility3.plugins.windows.mftscan*), 703
- minimum_address (*AVMLLayer* property), 183
- minimum_address (*BufferDataLayer* property), 226
- minimum_address (*DataLayerInterface* property), 155
- minimum_address (*Elf64Layer* property), 193
- minimum_address (*FileLayer* property), 229
- minimum_address (*Intel* attribute), 196
- minimum_address (*Intel32e* attribute), 199
- minimum_address (*IntelPAE* attribute), 202
- minimum_address (*LimeLayer* property), 215
- minimum_address (*LinearlyMappedLayer* property), 218
- minimum_address (*NonLinearlySegmentedLayer* property), 240
- minimum_address (*PdbMSFStream* property), 221
- minimum_address (*PdbMultiStreamFormat* property), 224
- minimum_address (*QemuSuspendLayer* property), 232
- minimum_address (*RegistryHive* property), 235
- minimum_address (*SegmentedLayer* property), 242
- minimum_address (*TranslationLayerInterface* property), 160
- minimum_address (*VmwareLayer* property), 245
- minimum_address (*WindowsCrashDump32Layer* property), 187
- minimum_address (*WindowsCrashDump64Layer* property), 189
- minimum_address (*WindowsIntel* attribute), 204
- minimum_address (*WindowsIntel32e* attribute), 207
- minimum_address (*WindowsIntelPAE* attribute), 210
- minimum_address (*WindowsMixin* attribute), 213
- minimum_address (*XenCoreDumpLayer* property), 248
- minor (*super_block* property), 376
- MINORBITS (*super_block* attribute), 374
- MissingModuleException, 556
- mm_struct (class in *volatility3.framework.symbols.linux.extensions*), 350
- mm_struct.VolTemplateProxy (class in *volatility3.framework.symbols.linux.extensions*), 351
- MMVAD (class in *volatility3.framework.symbols.windows.extensions*), 455
- MMVAD.VolTemplateProxy (class in *volatility3.framework.symbols.windows.extensions*), 455
- MMVAD_SHORT (class in *volatility3.framework.symbols.windows.extensions*), 457
- MMVAD_SHORT.VolTemplateProxy (class in *volatility3.framework.symbols.windows.extensions*), 458
- MNT_FLAGS (*mount* attribute), 357
- mnt_id (*MountInfoData* attribute), 592
- mnt_namespace (class in *volatility3.framework.symbols.linux.extensions*),

- 352
- `mnt_namespace.VolTemplateProxy` (class in `volatility3.framework.symbols.linux.extensions`), 353
- `MNT_NOATIME` (mount attribute), 357
- `MNT_NODEV` (mount attribute), 357
- `MNT_NODIRATIME` (mount attribute), 357
- `MNT_NOEXEC` (mount attribute), 357
- `MNT_NOSUID` (mount attribute), 357
- `mnt_opts` (`MountInfoData` attribute), 592
- `MNT_READONLY` (mount attribute), 357
- `MNT_RELATIME` (mount attribute), 357
- `mnt_root_path` (`MountInfoData` attribute), 592
- `MNT_SHARED` (mount attribute), 357
- `MNT_SHRINKABLE` (mount attribute), 357
- `mnt_type` (`MountInfoData` attribute), 592
- `MNT_UNBINDABLE` (mount attribute), 357
- `MNT_WRITE_HOLD` (mount attribute), 357
- `MODIFIED` (`TimeLinerType` attribute), 762
- `ModScan` (class in `volatility3.plugins.windows.modscan`), 705
- module
 - `volatility3`, 47
 - `volatility3.cli`, 48
 - `volatility3.cli.text_renderer`, 64
 - `volatility3.cli.volargparse`, 67
 - `volatility3.cli.volshell`, 49
 - `volatility3.cli.volshell.generic`, 50
 - `volatility3.cli.volshell.linux`, 55
 - `volatility3.cli.volshell.mac`, 58
 - `volatility3.cli.volshell.windows`, 61
 - `volatility3.framework`, 68
 - `volatility3.framework.automagic`, 69
 - `volatility3.framework.automagic.construct_layers`, 70
 - `volatility3.framework.automagic.linux`, 72
 - `volatility3.framework.automagic.mac`, 74
 - `volatility3.framework.automagic.module`, 77
 - `volatility3.framework.automagic.pdbscan`, 78
 - `volatility3.framework.automagic.stacker`, 82
 - `volatility3.framework.automagic.symbol_cache`, 85
 - `volatility3.framework.automagic.symbol_finder`, 90
 - `volatility3.framework.automagic.windows`, 92
 - `volatility3.framework.configuration`, 95
 - `volatility3.framework.configuration.requirements`, 95
 - `volatility3.framework.constants`, 119
 - `volatility3.framework.constants.linux`, 122
 - `volatility3.framework.constants.windows`, 122
 - `volatility3.framework.contexts`, 123
 - `volatility3.framework.exceptions`, 555
 - `volatility3.framework.interfaces`, 134
 - `volatility3.framework.interfaces.automagic`, 134
 - `volatility3.framework.interfaces.configuration`, 137
 - `volatility3.framework.interfaces.context`, 148
 - `volatility3.framework.interfaces.layers`, 154
 - `volatility3.framework.interfaces.objects`, 161
 - `volatility3.framework.interfaces.plugins`, 165
 - `volatility3.framework.interfaces.renderers`, 168
 - `volatility3.framework.interfaces.symbols`, 172
 - `volatility3.framework.layers`, 180
 - `volatility3.framework.layers.avml`, 182
 - `volatility3.framework.layers.cloudstorage`, 185
 - `volatility3.framework.layers.codecs`, 180
 - `volatility3.framework.layers.crash`, 185
 - `volatility3.framework.layers.elf`, 191
 - `volatility3.framework.layers.intel`, 195
 - `volatility3.framework.layers.leechcore`, 214
 - `volatility3.framework.layers.lime`, 214
 - `volatility3.framework.layers.linear`, 217
 - `volatility3.framework.layers.msf`, 220
 - `volatility3.framework.layers.physical`, 225
 - `volatility3.framework.layers.qemu`, 230
 - `volatility3.framework.layers.registry`, 233
 - `volatility3.framework.layers.resources`, 237
 - `volatility3.framework.layers.scanners`, 180
 - `volatility3.framework.layers.scanners.multiregexp`, 181
 - `volatility3.framework.layers.segmented`, 239
 - `volatility3.framework.layers.vmware`, 243
 - `volatility3.framework.layers.xen`, 246
 - `volatility3.framework.objects`, 249
 - `volatility3.framework.objects.templates`, 298
 - `volatility3.framework.objects.utility`, 300

volatility3.framework.plugins, 301
 volatility3.framework.renderers, 301
 volatility3.framework.renderers.conversion, 304
 volatility3.framework.renderers.format_hints, 305
 volatility3.framework.symbols, 316
 volatility3.framework.symbols.generic, 319
 volatility3.framework.symbols.intermed, 523
 volatility3.framework.symbols.linux, 321
 volatility3.framework.symbols.linux.bash, 397
 volatility3.framework.symbols.linux.extensions, 326
 volatility3.framework.symbols.linux.extensions.volatile, 390
 volatility3.framework.symbols.linux.extensions.elf, 392
 volatility3.framework.symbols.mac, 401
 volatility3.framework.symbols.mac.extensions, 405
 volatility3.framework.symbols.metadata, 553
 volatility3.framework.symbols.native, 553
 volatility3.framework.symbols.windows, 429
 volatility3.framework.symbols.windows.extensions, 432
 volatility3.framework.symbols.windows.extensions.crash, 470
 volatility3.framework.symbols.windows.extensions.kdhi, 472
 volatility3.framework.symbols.windows.extensions.kdhi, 474
 volatility3.framework.symbols.windows.extensions.kdhi, 479
 volatility3.framework.symbols.windows.extensions.kdhi, 485
 volatility3.framework.symbols.windows.extensions.kdhi, 485
 volatility3.framework.symbols.windows.extensions.kdhi, 489
 volatility3.framework.symbols.windows.extensions.kdhi, 500
 volatility3.framework.symbols.windows.extensions.kdhi, 512
 volatility3.framework.symbols.windows.extensions.kdhi, 517
 volatility3.framework.symbols.windows.extensions.kdhi, 519
 volatility3.framework.symbols.windows.versions, 522
 volatility3.framework.symbols.wrappers, 555
 volatility3.plugins, 558
 volatility3.plugins.banners, 753
 volatility3.plugins.configwriter, 755
 volatility3.plugins.frameworkinfo, 757
 volatility3.plugins.isfinfo, 758
 volatility3.plugins.layerwriter, 760
 volatility3.plugins.linux, 558
 volatility3.plugins.linux.bash, 559
 volatility3.plugins.linux.capabilities, 560
 volatility3.plugins.linux.check_afinfo, 563
 volatility3.plugins.linux.check_creds, 565
 volatility3.plugins.linux.check_idt, 566
 volatility3.plugins.linux.check_modules, 568
 volatility3.plugins.linux.check_syscall, 569
 volatility3.plugins.linux.elfs, 571
 volatility3.plugins.linux.envvars, 573
 volatility3.plugins.linux.iomem, 575
 volatility3.plugins.linux.keyboard_notifiers, 577
 volatility3.plugins.linux.kmsg, 578
 volatility3.plugins.linux.lsmmod, 585
 volatility3.plugins.linux.lsof, 587
 volatility3.plugins.linux.malfind, 588
 volatility3.plugins.linux.mountinfo, 590
 volatility3.plugins.linux.proc, 592
 volatility3.plugins.linux.psaux, 595
 volatility3.plugins.linux.pslist, 596
 volatility3.plugins.linux.psscan, 599
 volatility3.plugins.linux.pstree, 601
 volatility3.plugins.linux.sockstat, 602
 volatility3.plugins.linux.tty_check, 605
 volatility3.plugins.linux.vmayarascan, 606
 volatility3.plugins.mac, 608
 volatility3.plugins.mac.bash, 608
 volatility3.plugins.mac.check_syscall, 610
 volatility3.plugins.mac.check_sysctl, 612
 volatility3.plugins.mac.check_trap_table, 615
 volatility3.plugins.mac.ifconfig, 615
 volatility3.plugins.mac.kauth_listeners, 616
 volatility3.plugins.mac.kauth_scopes, 618
 volatility3.plugins.mac.kevents, 620
 volatility3.plugins.mac.list_files, 622
 volatility3.plugins.mac.lsmmod, 623

- volatility3.plugins.mac.lsof, 625
- volatility3.plugins.mac.malfind, 627
- volatility3.plugins.mac.mount, 628
- volatility3.plugins.mac.netstat, 630
- volatility3.plugins.mac.proc_maps, 632
- volatility3.plugins.mac.psaux, 633
- volatility3.plugins.mac.pslist, 635
- volatility3.plugins.mac.pstree, 639
- volatility3.plugins.mac.socket_filters, 640
- volatility3.plugins.mac.timers, 642
- volatility3.plugins.mac.trustedbsd, 643
- volatility3.plugins.mac.vfsevents, 645
- volatility3.plugins.timeliner, 762
- volatility3.plugins.windows, 647
- volatility3.plugins.windows.bigpools, 655
- volatility3.plugins.windows.cachedump, 657
- volatility3.plugins.windows.callbacks, 659
- volatility3.plugins.windows.cmdline, 662
- volatility3.plugins.windows.crashinfo, 664
- volatility3.plugins.windows.devicetree, 666
- volatility3.plugins.windows.dllexport, 667
- volatility3.plugins.windows.driverirp, 669
- volatility3.plugins.windows.drivermodule, 671
- volatility3.plugins.windows.driverscan, 673
- volatility3.plugins.windows.dumpfiles, 675
- volatility3.plugins.windows.envvars, 677
- volatility3.plugins.windows.filescan, 678
- volatility3.plugins.windows.getservicesidsmodule, 680
- volatility3.plugins.windows.getsids, 682
- volatility3.plugins.windows.handles, 684
- volatility3.plugins.windows.hashdump, 686
- volatility3.plugins.windows.info, 689
- volatility3.plugins.windows.joblinks, 691
- volatility3.plugins.windows.ldrmodules, 693
- volatility3.plugins.windows.lsadump, 694
- volatility3.plugins.windows.malfind, 696
- volatility3.plugins.windows.mbrscan, 698
- volatility3.plugins.windows.memmap, 700
- volatility3.plugins.windows.mftscan, 702
- volatility3.plugins.windows.modscan, 705
- volatility3.plugins.windows.modules, 707
- volatility3.plugins.windows.mutantscan, 710
- volatility3.plugins.windows.netscan, 712
- volatility3.plugins.windows.netstat, 714
- volatility3.plugins.windows.poolscanner, 719
- volatility3.plugins.windows.privileges, 724
- volatility3.plugins.windows.pslist, 726
- volatility3.plugins.windows.psscan, 729
- volatility3.plugins.windows.pstree, 731
- volatility3.plugins.windows.registry, 647
- volatility3.plugins.windows.registry.hivelist, 647
- volatility3.plugins.windows.registry.hivescan, 650
- volatility3.plugins.windows.registry.printkey, 652
- volatility3.plugins.windows.registry.userassist, 654
- volatility3.plugins.windows.sessions, 733
- volatility3.plugins.windows.skeleton_key_check, 734
- volatility3.plugins.windows.ssdt, 736
- volatility3.plugins.windows.strings, 738
- volatility3.plugins.windows.svcscan, 740
- volatility3.plugins.windows.symlinkscan, 742
- volatility3.plugins.windows.vadinfo, 744
- volatility3.plugins.windows.vadwalk, 746
- volatility3.plugins.windows.vadyarascan, 748
- volatility3.plugins.windows.verinfo, 750
- volatility3.plugins.windows.virtmap, 752
- volatility3.plugins.yarascan, 765
- volatility3.schemas, 767
- volatility3.symbols, 768
- Module (class in volatility3.framework.contexts), 127
- Module (class in volatility3.framework.symbols.linux.extensions), 354
- module() (Context method), 126
- module() (ContextInterface method), 149
- module.VolTemplateProxy (class in volatility3.framework.symbols.linux.extensions), 354
- module_from_pdb() (PDBUtility class method), 520
- ModuleCollection (class in volatility3.framework.contexts), 130
- ModuleContainer (class in volatility3.framework.interfaces.context), 150
- ModuleInterface (class in volatility3.framework.interfaces.context), 150
- ModuleRequirement (class in volatility3.framework.configuration.requirements), 106

- Modules (class in *volatility3.plugins.windows.modules*), 707
- modules (Context property), 127
- modules (ContextInterface property), 149
- modules (ModuleCollection property), 131
- mount (class in *volatility3.framework.symbols.linux.extensions*), 357
- Mount (class in *volatility3.plugins.mac.mount*), 628
- mount.VolTemplateProxy (class in *volatility3.framework.symbols.linux.extensions*), 357
- MountInfo (class in *volatility3.plugins.linux.mountinfo*), 590
- MountInfoData (class in *volatility3.plugins.linux.mountinfo*), 591
- MT_FLAGS_HEIGHT_MASK (maple_tree attribute), 349
- MT_FLAGS_HEIGHT_OFFSET (maple_tree attribute), 349
- Multiprocessing (Parallelism attribute), 120
- MultiRegex (class in *volatility3.framework.layers.scanners.multiregexp*), 181
- MultiRequirement (class in *volatility3.framework.configuration.requirements*), 108
- MultiStringScanner (class in *volatility3.framework.layers.scanners*), 180
- MultiTypeData (class in *volatility3.framework.renderers.format_hints*), 312
- multitypedata_as_text() (in module *volatility3.cli.text_renderer*), 67
- MutantScan (class in *volatility3.plugins.windows.mutantscan*), 710
- MuteProgress (class in *volatility3.cli*), 49
- N**
- name (AVMLLayer property), 183
- name (BooleanRequirement property), 96
- name (BufferDataLayer property), 226
- name (BytesRequirement property), 97
- name (ChoiceRequirement property), 99
- name (ClassRequirement property), 138
- name (CLIRenderer attribute), 64
- name (CMHIVE property), 502
- name (Column attribute), 169
- name (ComplexListRequirement property), 100
- name (ConfigurableModule property), 124
- name (ConfigurableRequirementInterface property), 140
- name (ConstructableRequirementInterface property), 142
- name (CSVRenderer attribute), 64
- name (DataLayerInterface property), 155
- name (Elf64Layer property), 193
- name (FileLayer property), 229
- name (Intel property), 196
- name (Intel32e property), 199
- name (IntelPAE property), 202
- name (IntRequirement property), 102
- name (JsonLinesRenderer attribute), 65
- name (JsonRenderer attribute), 65
- name (LayerListRequirement property), 103
- name (LimeLayer property), 215
- name (LinearlyMappedLayer property), 219
- name (ListRequirement property), 105
- name (Module property), 129
- name (ModuleInterface property), 152
- name (ModuleRequirement property), 107
- name (MultiRequirement property), 108
- name (NoneRenderer attribute), 65
- name (NonLinearlySegmentedLayer property), 240
- name (PdbMSFStream property), 221
- name (PdbMultiStreamFormat property), 224
- name (PluginRequirement property), 110
- name (PrettyTextRenderer attribute), 66
- name (QemuSuspendLayer property), 232
- name (QuickTextRenderer attribute), 66
- name (RegistryHive property), 235
- name (RequirementInterface property), 145
- name (SegmentedLayer property), 242
- name (SimpleTypeRequirement property), 147
- name (SizedModule property), 133
- name (StringRequirement property), 112
- name (SymbolInterface property), 176
- name (SymbolTableRequirement property), 113
- name (TranslationLayerInterface property), 160
- name (TranslationLayerRequirement property), 115
- name (URIRequirement property), 117
- name (VersionRequirement property), 118
- name (VmwareLayer property), 245
- name (WindowsCrashDump32Layer property), 187
- name (WindowsCrashDump64Layer property), 189
- name (WindowsIntel property), 204
- name (WindowsIntel32e property), 207
- name (WindowsIntelPAE property), 210
- name (WindowsMixin property), 213
- name (XenCoreDumpLayer property), 248
- name_as_str() (qstr method), 368
- name_strip() (PdbReader method), 518
- NameInfo (OBJECT_HEADER property), 491
- natives (BaseSymbolTableInterface property), 173
- natives (BashIntermedSymbols property), 400
- natives (IntermediateSymbolTable property), 528
- natives (ISFormatTable property), 525
- natives (LinuxKernelIntermedSymbols property), 323
- natives (MacKernelIntermedSymbols property), 403
- natives (NativeTable property), 554
- natives (NativeTableInterface property), 175
- natives (SymbolTableInterface property), 179

- natives (*Version1Format* property), 531
- natives (*Version2Format* property), 534
- natives (*Version3Format* property), 537
- natives (*Version4Format* property), 540
- natives (*Version5Format* property), 543
- natives (*Version6Format* property), 546
- natives (*Version7Format* property), 549
- natives (*Version8Format* property), 552
- natives (*WindowsKernelIntermedSymbols* property), 432
- NativeTable (class in *volatility3.framework.symbols.native*), 553
- NativeTableInterface (class in *volatility3.framework.interfaces.symbols*), 174
- net (class in *volatility3.framework.symbols.linux.extensions*), 360
- net.VolTemplateProxy (class in *volatility3.framework.symbols.linux.extensions*), 361
- netlink_sock (class in *volatility3.framework.symbols.linux.extensions*), 362
- netlink_sock.VolTemplateProxy (class in *volatility3.framework.symbols.linux.extensions*), 362
- NetScan (class in *volatility3.plugins.windows.netscan*), 712
- Netstat (class in *volatility3.plugins.mac.netstat*), 630
- NetStat (class in *volatility3.plugins.windows.netstat*), 714
- new_requirement() (*ComplexListRequirement* method), 100
- new_requirement() (*LayerListRequirement* method), 103
- next_peer() (*mount* method), 360
- non_cached_schemes() (*JarHandler* class method), 237
- non_cached_schemes() (*OfflineHandler* class method), 237
- non_cached_schemes() (*VolatilityHandler* class method), 238
- NoneRenderer (class in *volatility3.cli.text_renderer*), 65
- NonInheritable (class in *volatility3.framework*), 68
- NonLinearlySegmentedLayer (class in *volatility3.framework.layers.segmented*), 239
- NONPAGED (*PoolType* attribute), 722
- NotApplicableValue (class in *volatility3.framework.renderers*), 301
- NotAvailableValue (class in *volatility3.framework.renderers*), 301
- nsec_to_sec_str() (*ABCKmsg* method), 579
- nsec_to_sec_str() (*KmsgFiveTen* method), 582
- nsec_to_sec_str() (*KmsgLegacy* method), 584
- NullFileHandler (class in *volatility3.cli.volshell.generic*), 50
- num_symtab (*module* property), 357
- numerator (*Bin* attribute), 306
- numerator (*BitField* attribute), 255
- numerator (*Boolean* attribute), 258
- numerator (*Char* attribute), 267
- numerator (*Enumeration* attribute), 273
- numerator (*Hex* attribute), 308
- numerator (*Integer* attribute), 280
- numerator (*Parallelism* attribute), 121
- numerator (*Pointer* attribute), 283
- numerator (*PoolType* attribute), 723
- numerator (*RegKeyFlags* attribute), 511
- numerator (*TimeLinerType* attribute), 763
- object() (*ConfigurableModule* method), 124
- object() (*Context* method), 127
- object() (*ContextInterface* method), 149
- object() (*Module* method), 129
- object() (*ModuleInterface* method), 152
- object() (*SizedModule* method), 133
- object_from_symbol() (*ConfigurableModule* method), 125
- object_from_symbol() (*Module* method), 129
- object_from_symbol() (*ModuleInterface* method), 153
- object_from_symbol() (*SizedModule* method), 133
- OBJECT_HEADER (class in *volatility3.framework.symbols.windows.extensions.pool*), 491
- OBJECT_HEADER.VolTemplateProxy (class in *volatility3.framework.symbols.windows.extensions.pool*), 491
- OBJECT_SYMBOLIC_LINK (class in *volatility3.framework.symbols.windows.extensions*), 460
- OBJECT_SYMBOLIC_LINK.VolTemplateProxy (class in *volatility3.framework.symbols.windows.extensions*), 460
- ObjectInformation (class in *volatility3.framework.interfaces.objects*), 161
- ObjectInterface (class in *volatility3.framework.interfaces.objects*), 161
- ObjectInterface.VolTemplateProxy (class in *volatility3.framework.interfaces.objects*), 162
- ObjectTemplate (class in *volatility3.framework.objects.templates*), 298
- odd_parity (*Hashdump* attribute), 688
- Off (*Parallelism* attribute), 120
- OFFLINE (in *module volatility3.framework.constants*), 120
- OfflineException, 556

OfflineHandler (class in volatility3.framework.layers.resources), 237
 offset (ConfigurableModule property), 125
 offset (Module property), 129
 offset (ModuleInterface property), 153
 offset (SizedModule property), 133
 omap_lookup() (PdbReader method), 518
 open (ADS property), 702
 open (Banners property), 754
 open (Bash property), 560, 609
 open (BigPools property), 656
 open (Cachedump property), 658
 open (Callbacks property), 661
 open (Capabilities property), 562
 open (Check_afinfo property), 564
 open (Check_creds property), 565
 open (Check_idt property), 567
 open (Check_modules property), 569
 open (Check_syscall property), 570, 611
 open (Check_sysctl property), 612
 open (Check_trap_table property), 614
 open (CmdLine property), 663
 open (ConfigWriter property), 756
 open (Crashinfo property), 665
 open (DeviceTree property), 666
 open (DllList property), 668
 open (DriverIrp property), 670
 open (DriverModule property), 672
 open (DriverScan property), 674
 open (DumpFiles property), 676
 open (Elfs property), 572
 open (Envvars property), 574, 678
 open (FileScan property), 679
 open (FrameworkInfo property), 757
 open (GetServiceSIDs property), 681
 open (GetSIDs property), 683
 open (Handles property), 685
 open (Hashdump property), 688
 open (HiveList property), 649
 open (HiveScan property), 650
 open (Ifconfig property), 616
 open (Info property), 690
 open (IOMem property), 575
 open (IsfInfo property), 759
 open (JobLinks property), 692
 open (Kauth_listeners property), 617
 open (Kauth_scopes property), 619
 open (Kevents property), 621
 open (Keyboard_notifiers property), 577
 open (Kmsg property), 580
 open (LayerWriter property), 761
 open (LdrModules property), 694
 open (List_Files property), 623
 open (Lsadbump property), 695
 open (Lsmod property), 586, 624
 open (Lsof property), 587, 626
 open (Malfind property), 589, 627, 698
 open (Maps property), 593, 633
 open (MBRScan property), 699
 open (Memmap property), 701
 open (MFTScan property), 704
 open (ModScan property), 706
 open (Modules property), 709
 open (Mount property), 629
 open (MountInfo property), 591
 open (MutantScan property), 710
 open (NetScan property), 713
 open (NetStat property), 717
 open (Netstat property), 631
 open (PluginInterface property), 168
 open (PoolScanner property), 721
 open (PrintKey property), 653
 open (Privs property), 725
 open (PsAux property), 595
 open (Psaux property), 634
 open (PsList property), 598, 638, 727
 open (PsScan property), 600, 730
 open (PsTree property), 602, 639, 732
 open (Sessions property), 734
 open (Skeleton_Key_Check property), 735
 open (Socket_filters property), 641
 open (Sockstat property), 604
 open (SSDT property), 737
 open (Strings property), 739
 open (SvcScan property), 741
 open (SymlinkScan property), 743
 open (Timeliner property), 765
 open (Timers property), 643
 open (Trustedbsd property), 644
 open (tty_check property), 605
 open (UserAssist property), 655
 open (VadInfo property), 745
 open (VadWalk property), 747
 open (VadYaraScan property), 749
 open (VerInfo property), 751
 open (VFSevents property), 646
 open (VirtMap property), 753
 open (VmaYaraScan property), 607
 open (Volshell property), 54, 57, 60, 63
 open (YaraScan property), 766
 open() (ResourceAccessor method), 238
 operating_system (IdentifierProcessor attribute), 86
 operating_system (LinuxIdentifier attribute), 86
 operating_system (LinuxSymbolFinder attribute), 74
 operating_system (MacIdentifier attribute), 87
 operating_system (MacSymbolFinder attribute), 76
 operating_system (SymbolFinder attribute), 92
 operating_system (WindowsIdentifier attribute), 90

- [optional \(*BooleanRequirement* property\), 96](#)
 - [optional \(*BytesRequirement* property\), 97](#)
 - [optional \(*ChoiceRequirement* property\), 99](#)
 - [optional \(*ClassRequirement* property\), 138](#)
 - [optional \(*ComplexListRequirement* property\), 100](#)
 - [optional \(*ConfigurableRequirementInterface* property\), 140](#)
 - [optional \(*ConstructableRequirementInterface* property\), 142](#)
 - [optional \(*IntRequirement* property\), 102](#)
 - [optional \(*LayerListRequirement* property\), 104](#)
 - [optional \(*ListRequirement* property\), 105](#)
 - [optional \(*ModuleRequirement* property\), 107](#)
 - [optional \(*MultiRequirement* property\), 108](#)
 - [optional \(*PluginRequirement* property\), 110](#)
 - [optional \(*RequirementInterface* property\), 145](#)
 - [optional \(*SimpleTypeRequirement* property\), 147](#)
 - [optional \(*StringRequirement* property\), 112](#)
 - [optional \(*SymbolTableRequirement* property\), 113](#)
 - [optional \(*TranslationLayerRequirement* property\), 115](#)
 - [optional \(*URIRequirement* property\), 117](#)
 - [optional \(*VersionRequirement* property\), 118](#)
 - [optional\(\) \(in module *volatility3.cli.text_renderer*\), 67](#)
 - [optional_set_type_class\(\) \(*BaseSymbolTableInterface* method\), 173](#)
 - [optional_set_type_class\(\) \(*BashIntermedSymbols* method\), 400](#)
 - [optional_set_type_class\(\) \(*IntermediateSymbolTable* method\), 528](#)
 - [optional_set_type_class\(\) \(*ISFormatTable* method\), 525](#)
 - [optional_set_type_class\(\) \(*LinuxKernelIntermedSymbols* method\), 323](#)
 - [optional_set_type_class\(\) \(*MacKernelIntermedSymbols* method\), 403](#)
 - [optional_set_type_class\(\) \(*NativeTable* method\), 555](#)
 - [optional_set_type_class\(\) \(*NativeTableInterface* method\), 175](#)
 - [optional_set_type_class\(\) \(*SymbolTableInterface* method\), 179](#)
 - [optional_set_type_class\(\) \(*Version1Format* method\), 531](#)
 - [optional_set_type_class\(\) \(*Version2Format* method\), 534](#)
 - [optional_set_type_class\(\) \(*Version3Format* method\), 537](#)
 - [optional_set_type_class\(\) \(*Version4Format* method\), 540](#)
 - [optional_set_type_class\(\) \(*Version5Format* method\), 543](#)
 - [optional_set_type_class\(\) \(*Version6Format* method\), 546](#)
 - [optional_set_type_class\(\) \(*Version7Format* method\), 549](#)
 - [optional_set_type_class\(\) \(*Version8Format* method\), 552](#)
 - [optional_set_type_class\(\) \(*WindowsKernelIntermedSymbols* method\), 432](#)
 - [OsDistinguisher \(class in *volatility3.framework.symbols.windows.versions*\), 522](#)
 - [output_result\(\) \(*JsonLinesRenderer* method\), 65](#)
 - [output_result\(\) \(*JsonRenderer* method\), 65](#)
 - [overlap \(*PageMapScanner* attribute\), 93](#)
 - [overlap \(*PdbSignatureScanner* attribute\), 522](#)
 - [owning_process\(\) \(*ETHREAD* method\), 443](#)
- ## P
- [PACKAGE_VERSION \(in module *volatility3.framework.constants*\), 120](#)
 - [packet_sock \(class in *volatility3.framework.symbols.linux.extensions*\), 364](#)
 - [packet_sock.VolTemplateProxy \(class in *volatility3.framework.symbols.linux.extensions*\), 364](#)
 - [PAGE_MASK \(*CONTROL_AREA* attribute\), 433](#)
 - [PAGE_SHIFT \(in module *volatility3.framework.constants.linux*\), 122](#)
 - [PAGE_SIZE \(*CONTROL_AREA* attribute\), 433](#)
 - [page_size \(*Intel* attribute\), 196](#)
 - [page_size \(*Intel32e* attribute\), 199](#)
 - [page_size \(*IntelPAE* attribute\), 202](#)
 - [page_size \(*PdbMultiStreamFormat* property\), 224](#)
 - [page_size \(*WindowsIntel* attribute\), 205](#)
 - [page_size \(*WindowsIntel32e* attribute\), 207](#)
 - [page_size \(*WindowsIntelPAE* attribute\), 210](#)
 - [page_size \(*WindowsMixin* attribute\), 213](#)
 - [PAGED \(*PoolType* attribute\), 722](#)
 - [PagedInvalidAddressException, 556](#)
 - [PageMapScanner \(class in *volatility3.framework.automagic.windows*\), 93](#)
 - [Parallelism \(class in *volatility3.framework.constants*\), 120](#)
 - [PARALLELISM \(in module *volatility3.framework.constants*\), 120](#)
 - [parent \(*TreeNode* property\), 171, 304](#)
 - [parent_e_type \(*elf_phdr* property\), 395](#)
 - [parent_id \(*MountInfoData* attribute\), 592](#)
 - [parent_offset \(*elf_phdr* property\), 395](#)
 - [parent_path\(\) \(in module *volatility3.framework.interfaces.configuration*\), 148](#)
 - [parse_args\(\) \(*HelpfulArgParser* method\), 68](#)
 - [parse_bitmap\(\) \(*NetStat* class method\), 717](#)
 - [parse_cache_entry\(\) \(*Cachedump* static method\), 658](#)
 - [parse_decrypted_cache\(\) \(*Cachedump* static method\), 658](#)

- [parse_hashtable\(\)](#) (*NetStat class method*), 717
[parse_intermixed_args\(\)](#) (*HelpfulArgParser method*), 68
[parse_known_args\(\)](#) (*HelpfulArgParser method*), 68
[parse_known_intermixed_args\(\)](#) (*HelpfulArgParser method*), 68
[parse_partitions\(\)](#) (*NetStat class method*), 718
[parse_resource\(\)](#) (*IOMem class method*), 575
[parse_string\(\)](#) (*PdbReader static method*), 518
[parse_userassist_data\(\)](#) (*UserAssist method*), 655
[partition\(\)](#) (*Bytes method*), 262
[partition\(\)](#) (*HexBytes method*), 310
[partition\(\)](#) (*MultiTypeData method*), 314
[partition\(\)](#) (*String method*), 290
[PARTITION_ENTRY](#) (class in *volatility3.framework.symbols.windows.extensions.mbr*), 474
[PARTITION_ENTRY.VolTemplateProxy](#) (class in *volatility3.framework.symbols.windows.extensions.mbr*), 474
[PARTITION_TABLE](#) (class in *volatility3.framework.symbols.windows.extensions.mbr*), 477
[PARTITION_TABLE.VolTemplateProxy](#) (class in *volatility3.framework.symbols.windows.extensions.mbr*), 477
[path](#) (*TreeNode property*), 171, 304
[path\(\)](#) (*dentry method*), 333
[path_changed\(\)](#) (*TreeNode method*), 171, 304
[path_depth](#) (*TreeNode property*), 171, 304
[path_depth\(\)](#) (in *module volatility3.framework.interfaces.configuration*), 148
[path_depth\(\)](#) (*TreeGrid static method*), 170, 302
[path_for_file\(\)](#) (*LinuxUtilities class method*), 325
[path_head\(\)](#) (in *module volatility3.framework.interfaces.configuration*), 148
[path_join\(\)](#) (in *module volatility3.framework.interfaces.configuration*), 148
[path_root](#) (*MountInfoData attribute*), 592
[path_sep](#) (*TreeGrid attribute*), 302
[pci_hole_table](#) (*QemuSuspendLayer attribute*), 232
[pdb_age](#) (*WindowsMetadata property*), 553
[pdb_guid](#) (*WindowsMetadata property*), 553
[pdb_layer_name](#) (*PdbReader property*), 518
[pdb_symbol_table](#) (*PdbMSFStream property*), 221
[pdb_symbol_table](#) (*PdbMultiStreamFormat property*), 224
[PDBFormatException](#), 220
[PdbMSFStream](#) (class in *volatility3.framework.layers.msf*), 220
[PdbMultiStreamFormat](#) (class in *volatility3.framework.layers.msf*), 222
[pdbname_scan\(\)](#) (*PDBUtility class method*), 520
[PdbReader](#) (class in *volatility3.framework.symbols.windows.pdbconv*), 517
[PdbRetriever](#) (class in *volatility3.framework.symbols.windows.pdbconv*), 519
[PdbSignatureScanner](#) (class in *volatility3.framework.symbols.windows.pdbutil*), 521
[PDBUtility](#) (class in *volatility3.framework.symbols.windows.pdbutil*), 519
[pe_version](#) (*WindowsMetadata property*), 553
[pe_version_string](#) (*WindowsMetadata property*), 553
[perm_flags](#) (*vm_area_struct attribute*), 386
[PHYSICAL_DEFAULT](#) (*PsList attribute*), 726
[pid](#) (*TaskData attribute*), 563
[PluginInterface](#) (class in *volatility3.framework.interfaces.plugins*), 167
[PluginRequirement](#) (class in *volatility3.framework.configuration.requirements*), 109
[PluginRequirementException](#), 556
[PLUGINS_PATH](#) (in *module volatility3.framework.constants*), 120
[PluginVersionException](#), 557
[Pointer](#) (class in *volatility3.framework.objects*), 281
[Pointer.VolTemplateProxy](#) (class in *volatility3.framework.objects*), 281
[pointer_to_string\(\)](#) (in *module volatility3.framework.objects.utility*), 300
[POOL_HEADER](#) (class in *volatility3.framework.symbols.windows.extensions.pool*), 493
[POOL_HEADER.VolTemplateProxy](#) (class in *volatility3.framework.symbols.windows.extensions.pool*), 493
[POOL_HEADER_VISTA](#) (class in *volatility3.framework.symbols.windows.extensions.pool*), 495
[POOL_HEADER_VISTA.VolTemplateProxy](#) (class in *volatility3.framework.symbols.windows.extensions.pool*), 496
[pool_scan\(\)](#) (*PoolScanner class method*), 721
[POOL_TRACKER_BIG_PAGES](#) (class in *volatility3.framework.symbols.windows.extensions.pool*), 498
[POOL_TRACKER_BIG_PAGES.VolTemplateProxy](#) (class in *volatility3.framework.symbols.windows.extensions.pool*), 498

ity3.framework.symbols.windows.extensions.pool),
498

pool_type_lookup (*POOL_TRACKER_BIG_PAGES* attribute), 500

PoolConstraint (class in *volatility3.plugins.windows.poolscanner*), 719

PoolHeaderScanner (class in *volatility3.plugins.windows.poolscanner*), 719

PoolScanner (class in *volatility3.plugins.windows.poolscanner*), 719

PoolType (class in *volatility3.plugins.windows.poolscanner*), 722

populate() (*TreeGrid* method), 170, 302

populate_config() (*CommandLine* method), 48

populate_config() (*VolShell* method), 49

populate_requirements_argparse() (*CommandLine* method), 48

populate_requirements_argparse() (*VolShell* method), 50

populated (*TreeGrid* property), 170, 303

possible_architectures (*Disassembly* attribute), 169

ppid (*TaskData* attribute), 563

preferred_filename (*FileHandlerInterface* property), 165

preferred_filename (*NullFileHandler* property), 51

preprocess() (*MultiRegexp* method), 181

PrettyTextRenderer (class in *volatility3.cli.text_renderer*), 65

PrimitiveObject (class in *volatility3.framework.objects*), 284

PrimitiveObject.VolTemplateProxy (class in *volatility3.framework.objects*), 284

print_help() (*HelpfulArgParser* method), 68

print_usage() (*HelpfulArgParser* method), 68

PrintedProgress (class in *volatility3.cli*), 49

PrintKey (class in *volatility3.plugins.windows.registry.printkey*), 652

priority (*AutomagicInterface* attribute), 136

priority (*ConstructionMagic* attribute), 71

priority (*KernelModule* attribute), 78

priority (*KernelPDBScanner* attribute), 81

priority (*LayerStacker* attribute), 84

priority (*LinuxSymbolFinder* attribute), 74

priority (*MacSymbolFinder* attribute), 76

priority (*SymbolCacheMagic* attribute), 90

priority (*SymbolFinder* attribute), 92

priority (*WinSwapLayers* attribute), 94

privileges() (*TOKEN* method), 466

Privs (class in *volatility3.plugins.windows.privileges*), 724

proc (class in *volatility3.framework.symbols.mac.extensions*), 412

proc.VolTemplateProxy (class in *volatility3.framework.symbols.mac.extensions*), 412

proc_filters (*Kevents* attribute), 621

process() (*RemoteIdentifierFormat* method), 87

process_dump() (*PsList* class method), 727

process_exceptions() (*CommandLine* method), 49

process_exceptions() (*VolShell* method), 50

process_file_object() (*DumpFiles* class method), 676

process_index_array() (*SHARED_CACHE_MAP* method), 464

process_sock() (*SockHandlers* method), 602

process_types() (*PdbReader* method), 518

process_unsatisfied_exceptions() (*CommandLine* method), 49

process_unsatisfied_exceptions() (*VolShell* method), 50

process_v1() (*RemoteIdentifierFormat* method), 87

process_yara_options() (*YaraScan* class method), 766

ProgressCallback (in module *volatility3.framework.constants*), 122

protect_values() (*VadInfo* class method), 745

provides (*LinuxKernelIntermedSymbols* attribute), 323

provides (*MacKernelIntermedSymbols* attribute), 403

provides (*WindowsCrashDump32Layer* attribute), 187

provides (*WindowsCrashDump64Layer* attribute), 189

PsAux (class in *volatility3.plugins.linux.psaux*), 595

Psaux (class in *volatility3.plugins.mac.psaux*), 633

PsList (class in *volatility3.plugins.linux.pslist*), 596

PsList (class in *volatility3.plugins.mac.pslist*), 635

PsList (class in *volatility3.plugins.windows.pslist*), 726

pslist_methods (*PsList* attribute), 638

PSScan (class in *volatility3.plugins.linux.psscan*), 599

PSScan (class in *volatility3.plugins.windows.psscan*), 729

PSTree (class in *volatility3.plugins.linux.pstree*), 601

PSTree (class in *volatility3.plugins.mac.pstree*), 639

PSTree (class in *volatility3.plugins.windows.pstree*), 731

PYTHONPATH, 23

Q

QemuStacker (class in *volatility3.framework.layers.qemu*), 230

QemuSuspendLayer (class in *volatility3.framework.layers.qemu*), 230

QEVm_CONFIGURATION (*QemuSuspendLayer* attribute), 230

QEVm_EOF (*QemuSuspendLayer* attribute), 230

QEVm_SECTION_END (*QemuSuspendLayer* attribute), 230

QEVm_SECTION_FOOTER (*QemuSuspendLayer* attribute), 230

QEVm_SECTION_FULL (*QemuSuspendLayer* attribute), 230

- QEMV_SECTION_PART (*QemuSuspendLayer* attribute), 230
- QEMV_SECTION_START (*QemuSuspendLayer* attribute), 230
- QEMV_SUBSECTION (*QemuSuspendLayer* attribute), 230
- QEMV_VMDESCRIPTION (*QemuSuspendLayer* attribute), 231
- qstr (class in *volatility3.framework.symbols.linux.extensions*), 366
- qstr.VolTemplateProxy (class in *volatility3.framework.symbols.linux.extensions*), 366
- queue_entry (class in *volatility3.framework.symbols.mac.extensions*), 414
- queue_entry.VolTemplateProxy (class in *volatility3.framework.symbols.mac.extensions*), 414
- QuickTextRenderer (class in *volatility3.cli.text_renderer*), 66
- quoted_optional() (in module *volatility3.cli.text_renderer*), 67
- read_dbf_stream() (*PdbReader* method), 518
- read_ipi_stream() (*PdbReader* method), 518
- read_necessary_streams() (*PdbReader* method), 518
- read_pdb_info_stream() (*PdbReader* method), 518
- read_pointer() (*NetStat* class method), 718
- read_streams() (*PdbMultiStreamFormat* method), 224
- read_symbol_stream() (*PdbReader* method), 519
- read_tpi_stream() (*PdbReader* method), 519
- readable() (*FileHandlerInterface* method), 165
- readable() (*NullFileHandler* method), 51
- readall() (*FileHandlerInterface* method), 165
- readall() (*NullFileHandler* method), 51
- readinto() (*FileHandlerInterface* method), 165
- readinto() (*NullFileHandler* method), 51
- readinto1() (*NullFileHandler* method), 51
- readline() (*FileHandlerInterface* method), 165
- readline() (*NullFileHandler* method), 51
- readlines() (*FileHandlerInterface* method), 166
- readlines() (*NullFileHandler* method), 51
- ReadOnlyMapping (class in *volatility3.framework.interfaces.objects*), 163
- real (*Bin* attribute), 306
- real (*BitField* attribute), 255
- real (*Boolean* attribute), 258
- real (*Char* attribute), 267
- real (*Enumeration* attribute), 273
- real (*Float* attribute), 276
- real (*Hex* attribute), 308
- real (*Integer* attribute), 280
- real (*Parallelism* attribute), 121
- real (*Pointer* attribute), 284
- real (*PoolType* attribute), 724
- real (*RegKeyFlags* attribute), 511
- real (*TimeLinerType* attribute), 763
- reconstruct() (*IMAGE_DOS_HEADER* method), 487
- record_cached_validations() (in module *volatility3.schemas*), 767
- recurse_symbol_fulfiller() (*KernelPDBScanner* method), 81
- reference_count() (*kobject* method), 346
- ReferenceTemplate (class in *volatility3.framework.objects.templates*), 299
- REG_BINARY (*RegValueTypes* attribute), 512
- REG_DWORD (*RegValueTypes* attribute), 512
- REG_DWORD_BIG_ENDIAN (*RegValueTypes* attribute), 512
- REG_EXPAND_SZ (*RegValueTypes* attribute), 512
- REG_FULL_RESOURCE_DESCRIPTOR (*RegValueTypes* attribute), 512
- REG_LINK (*RegValueTypes* attribute), 512
- REG_MULTI_SZ (*RegValueTypes* attribute), 512
- REG_NONE (*RegValueTypes* attribute), 512
- REG_QWORD (*RegValueTypes* attribute), 512
- REG_RESOURCE_LIST (*RegValueTypes* attribute), 512
- random_string() (*Volshell* method), 54, 57, 60, 63
- read() (*AVMLLayer* method), 183
- read() (*BufferDataLayer* method), 226
- read() (*DataLayerInterface* method), 155
- read() (*Elf64Layer* method), 193
- read() (*FileHandlerInterface* method), 165
- read() (*FileLayer* method), 229
- read() (*Intel* method), 196
- read() (*Intel32e* method), 199
- read() (*IntelPAE* method), 202
- read() (*LayerContainer* method), 157
- read() (*LimeLayer* method), 215
- read() (*LinearlyMappedLayer* method), 219
- read() (*NonLinearlySegmentedLayer* method), 240
- read() (*NullFileHandler* method), 51
- read() (*PdbMSFStream* method), 221
- read() (*PdbMultiStreamFormat* method), 224
- read() (*QemuSuspendLayer* method), 232
- read() (*RegistryHive* method), 235
- read() (*SegmentedLayer* method), 242
- read() (*TranslationLayerInterface* method), 160
- read() (*VmwareLayer* method), 245
- read() (*WindowsCrashDump32Layer* method), 187
- read() (*WindowsCrashDump64Layer* method), 190
- read() (*WindowsIntel* method), 205
- read() (*WindowsIntel32e* method), 207
- read() (*WindowsIntelPAE* method), 210
- read() (*WindowsMixin* method), 213
- read() (*XenCoreDumpLayer* method), 248
- read1() (*NullFileHandler* method), 51

REG_RESOURCE_REQUIREMENTS_LIST (*RegValueTypes* attribute), 512
 REG_SZ (*RegValueTypes* attribute), 512
 REG_UNKNOWN (*RegValueTypes* attribute), 512
 RegExScanner (class in *volatility3.framework.layers.scanners*), 181
 register() (*HelpfulArgParser* method), 68
 RegistryFormatException, 233
 RegistryHive (class in *volatility3.framework.layers.registry*), 233
 RegistryInvalidIndex, 236
 RegKeyFlags (class in *volatility3.framework.symbols.windows.extensions.registry*), 510
 RegValueTypes (class in *volatility3.framework.symbols.windows.extensions.registry*), 512
 relative_child_offset() (*AggregateType.VolTemplateProxy* class method), 250
 relative_child_offset() (*Array.VolTemplateProxy* class method), 251
 relative_child_offset() (*BitField.VolTemplateProxy* class method), 253
 relative_child_offset() (*Boolean.VolTemplateProxy* class method), 256
 relative_child_offset() (*bpf_prog.VolTemplateProxy* class method), 326
 relative_child_offset() (*bt_sock.VolTemplateProxy* class method), 328
 relative_child_offset() (*Bytes.VolTemplateProxy* class method), 260
 relative_child_offset() (*Char.VolTemplateProxy* class method), 265
 relative_child_offset() (*ClassType.VolTemplateProxy* class method), 268
 relative_child_offset() (*CM_KEY_BODY.VolTemplateProxy* class method), 503
 relative_child_offset() (*CM_KEY_NODE.VolTemplateProxy* class method), 505
 relative_child_offset() (*CM_KEY_VALUE.VolTemplateProxy* class method), 507
 relative_child_offset() (*CMHIVE.VolTemplateProxy* class method), 501
 relative_child_offset() (*CONTROL_AREA.VolTemplateProxy* class method), 433
 relative_child_offset() (*cred.VolTemplateProxy* class method), 330
 relative_child_offset() (*dentry.VolTemplateProxy* class method), 332
 relative_child_offset() (*DEVICE_OBJECT.VolTemplateProxy* class method), 435
 relative_child_offset() (*DRIVER_OBJECT.VolTemplateProxy* class method), 437
 relative_child_offset() (*elf.VolTemplateProxy* class method), 392
 relative_child_offset() (*elf_phdr.VolTemplateProxy* class method), 394
 relative_child_offset() (*elf_sym.VolTemplateProxy* class method), 396
 relative_child_offset() (*Enumeration.VolTemplateProxy* class method), 270
 relative_child_offset() (*EPROCESS.VolTemplateProxy* class method), 439
 relative_child_offset() (*ETHREAD.VolTemplateProxy* class method), 442
 relative_child_offset() (*EX_FAST_REF.VolTemplateProxy* class method), 444
 relative_child_offset() (*ExecutiveObject.VolTemplateProxy* class method), 490
 relative_child_offset() (*FILE_OBJECT.VolTemplateProxy* class method), 446
 relative_child_offset() (*fileglob.VolTemplateProxy* class method), 405
 relative_child_offset() (*files_struct.VolTemplateProxy* class method), 334
 relative_child_offset() (*Float.VolTemplateProxy* class method), 274
 relative_child_offset() (*fs_struct.VolTemplateProxy* class method), 336
 relative_child_offset() (*Function.VolTemplateProxy* class method), 276
 relative_child_offset() (*GenericIntelProcess.VolTemplateProxy* class method), 320
 relative_child_offset() (*hist_entry.VolTemplateProxy* class method), 390
 relative_child_offset() (*HMAP_ENTRY.VolTemplateProxy* class method), 509

`relative_child_offset()` (*ifnet.VolTemplateProxy* class method), 407
`relative_child_offset()` (*IM-AGE_DOS_HEADER.VolTemplateProxy* class method), 485
`relative_child_offset()` (*IM-AGE_NT_HEADERS.VolTemplateProxy* class method), 488
`relative_child_offset()` (*inet_sock.VolTemplateProxy* class method), 338
`relative_child_offset()` (*inpcb.VolTemplateProxy* class method), 409
`relative_child_offset()` (*Integer.VolTemplateProxy* class method), 278
`relative_child_offset()` (*kauth_scope.VolTemplateProxy* class method), 411
`relative_child_offset()` (*KDDEBUG-GER_DATA64.VolTemplateProxy* class method), 473
`relative_child_offset()` (*kernel_cap_struct.VolTemplateProxy* class method), 340
`relative_child_offset()` (*kernel_cap_t.VolTemplateProxy* class method), 342
`relative_child_offset()` (*KMUTANT.VolTemplateProxy* class method), 448
`relative_child_offset()` (*kobject.VolTemplateProxy* class method), 345
`relative_child_offset()` (*KSYSTEM_TIME.VolTemplateProxy* class method), 450
`relative_child_offset()` (*KTHREAD.VolTemplateProxy* class method), 452
`relative_child_offset()` (*LIST_ENTRY.VolTemplateProxy* class method), 453
`relative_child_offset()` (*list_head.VolTemplateProxy* class method), 347
`relative_child_offset()` (*maple_tree.VolTemplateProxy* class method), 349
`relative_child_offset()` (*MFTAttribute.VolTemplateProxy* class method), 479
`relative_child_offset()` (*MFTEntry.VolTemplateProxy* class method), 481
`relative_child_offset()` (*MFTFileName.VolTemplateProxy* class method), 483
`relative_child_offset()` (*mm_struct.VolTemplateProxy* class method), 351
`relative_child_offset()` (*MM-VAD.VolTemplateProxy* class method), 455
`relative_child_offset()` (*MM-VAD_SHORT.VolTemplateProxy* class method), 458
`relative_child_offset()` (*mnt_namespace.VolTemplateProxy* class method), 353
`relative_child_offset()` (*module.VolTemplateProxy* class method), 355
`relative_child_offset()` (*mount.VolTemplateProxy* class method), 358
`relative_child_offset()` (*net.VolTemplateProxy* class method), 361
`relative_child_offset()` (*netlink_sock.VolTemplateProxy* class method), 363
`relative_child_offset()` (*OBJECT_HEADER.VolTemplateProxy* class method), 492
`relative_child_offset()` (*OBJECT_SYMBOLIC_LINK.VolTemplateProxy* class method), 460
`relative_child_offset()` (*ObjectInterface.VolTemplateProxy* class method), 162
`relative_child_offset()` (*ObjectTemplate* method), 299
`relative_child_offset()` (*packet_sock.VolTemplateProxy* class method), 365
`relative_child_offset()` (*PARTITION_ENTRY.VolTemplateProxy* class method), 475
`relative_child_offset()` (*PARTITION_TABLE.VolTemplateProxy* class method), 477
`relative_child_offset()` (*Pointer.VolTemplateProxy* class method), 281
`relative_child_offset()` (*POOL_HEADER.VolTemplateProxy* class method), 494
`relative_child_offset()` (*POOL_HEADER_VISTA.VolTemplateProxy* class method), 496
`relative_child_offset()` (*POOL_TRACKER_BIG_PAGES.VolTemplateProxy* class method), 498
`relative_child_offset()` (*PrimitiveObject.VolTemplateProxy* class method), 285
`relative_child_offset()` (*proc.VolTemplateProxy*

class method), 412
`relative_child_offset()` (*qstr.VolTemplateProxy class method*), 366
`relative_child_offset()` (*queue_entry.VolTemplateProxy class method*), 414
`relative_child_offset()` (*ReferenceTemplate method*), 300
`relative_child_offset()` (*SER-VICE_HEADER.VolTemplateProxy class method*), 513
`relative_child_offset()` (*SER-VICE_RECORD.VolTemplateProxy class method*), 515
`relative_child_offset()` (*SHARED_CACHE_MAP.VolTemplateProxy class method*), 463
`relative_child_offset()` (*sock.VolTemplateProxy class method*), 368
`relative_child_offset()` (*sock-addr.VolTemplateProxy class method*), 417
`relative_child_offset()` (*sock-addr_dl.VolTemplateProxy class method*), 418
`relative_child_offset()` (*socket.VolTemplateProxy class method*), 370, 420
`relative_child_offset()` (*String.VolTemplateProxy class method*), 286
`relative_child_offset()` (*struct_file.VolTemplateProxy class method*), 372
`relative_child_offset()` (*Struct-Type.VolTemplateProxy class method*), 293
`relative_child_offset()` (*SUM-MARY_DUMP.VolTemplateProxy class method*), 471
`relative_child_offset()` (*super_block.VolTemplateProxy class method*), 374
`relative_child_offset()` (*Symbol-Space.UnresolvedTemplate method*), 317
`relative_child_offset()` (*sysctl_oid.VolTemplateProxy class method*), 422
`relative_child_offset()` (*task_struct.VolTemplateProxy class method*), 376
`relative_child_offset()` (*Template method*), 164
`relative_child_offset()` (*TO-KEN.VolTemplateProxy class method*), 465
`relative_child_offset()` (*UNI-CODE_STRING.VolTemplateProxy class method*), 467
`relative_child_offset()` (*Union-Type.VolTemplateProxy class method*), 295
`relative_child_offset()` (*unix_sock.VolTemplateProxy class method*), 379
`relative_child_offset()` (*VACB.VolTemplateProxy class method*), 469
`relative_child_offset()` (*vfs-mount.VolTemplateProxy class method*), 381
`relative_child_offset()` (*vm_area_struct.VolTemplateProxy class method*), 384
`relative_child_offset()` (*vm_map_entry.VolTemplateProxy class method*), 424
`relative_child_offset()` (*vm_map_object.VolTemplateProxy class method*), 426
`relative_child_offset()` (*vnnode.VolTemplateProxy class method*), 428
`relative_child_offset()` (*Void.VolTemplateProxy class method*), 296
`relative_child_offset()` (*vsock_sock.VolTemplateProxy class method*), 386
`relative_child_offset()` (*xdp_sock.VolTemplateProxy class method*), 388
`REMOTE_ISF_URL` (*in module volatility3.framework.constants*), 122
`RemoteIdentifierFormat` (*class in volatility3.framework.automagic.symbol_cache*), 87
`remove()` (*SymbolSpace method*), 319
`remove_requirement()` (*BooleanRequirement method*), 96
`remove_requirement()` (*BytesRequirement method*), 97
`remove_requirement()` (*ChoiceRequirement method*), 99
`remove_requirement()` (*ClassRequirement method*), 138
`remove_requirement()` (*ComplexListRequirement method*), 101
`remove_requirement()` (*ConfigurableRequirementInterface method*), 140
`remove_requirement()` (*ConstructableRequirementInterface method*), 142
`remove_requirement()` (*IntRequirement method*), 102
`remove_requirement()` (*LayerListRequirement method*), 104
`remove_requirement()` (*ListRequirement method*), 105
`remove_requirement()` (*ModuleRequirement method*), 107

`remove_requirement()` (*MultiRequirement* method), 108
`remove_requirement()` (*PluginRequirement* method), 110
`remove_requirement()` (*RequirementInterface* method), 146
`remove_requirement()` (*SimpleTypeRequirement* method), 147
`remove_requirement()` (*StringRequirement* method), 112
`remove_requirement()` (*SymbolTableRequirement* method), 113
`remove_requirement()` (*TranslationLayerRequirement* method), 115
`remove_requirement()` (*URIRequirement* method), 117
`remove_requirement()` (*VersionRequirement* method), 118
`removeprefix()` (*Bytes* method), 263
`removeprefix()` (*HexBytes* method), 310
`removeprefix()` (*MultiTypeData* method), 314
`removeprefix()` (*String* method), 290
`removesuffix()` (*Bytes* method), 263
`removesuffix()` (*HexBytes* method), 310
`removesuffix()` (*MultiTypeData* method), 314
`removesuffix()` (*String* method), 290
`render()` (*CLIRenderer* method), 64
`render()` (*CSVRenderer* method), 64
`render()` (*JsonLinesRenderer* method), 65
`render()` (*JsonRenderer* method), 65
`render()` (*NoneRenderer* method), 65
`render()` (*PrettyTextRenderer* method), 66
`render()` (*QuickTextRenderer* method), 66
`render()` (*Renderer* method), 169
`render_treegrid()` (*Volshell* method), 54, 57, 60, 63
`Renderer` (class in *volatility3.framework.interfaces.renderers*), 169
`replace()` (*Bytes* method), 263
`replace()` (*HexBytes* method), 310
`replace()` (*MultiTypeData* method), 314
`replace()` (*String* method), 290
`replace_child()` (*AggregateType.VolTemplateProxy* class method), 250
`replace_child()` (*Array.VolTemplateProxy* class method), 251
`replace_child()` (*BitField.VolTemplateProxy* class method), 253
`replace_child()` (*Boolean.VolTemplateProxy* class method), 256
`replace_child()` (*bpf_prog.VolTemplateProxy* class method), 327
`replace_child()` (*bt_sock.VolTemplateProxy* class method), 328
`replace_child()` (*Bytes.VolTemplateProxy* class method), 260
`replace_child()` (*Char.VolTemplateProxy* class method), 265
`replace_child()` (*ClassType.VolTemplateProxy* class method), 268
`replace_child()` (*CM_KEY_BODY.VolTemplateProxy* class method), 503
`replace_child()` (*CM_KEY_NODE.VolTemplateProxy* class method), 505
`replace_child()` (*CM_KEY_VALUE.VolTemplateProxy* class method), 507
`replace_child()` (*CMHIVE.VolTemplateProxy* class method), 501
`replace_child()` (*CONTROL_AREA.VolTemplateProxy* class method), 433
`replace_child()` (*cred.VolTemplateProxy* class method), 330
`replace_child()` (*dentry.VolTemplateProxy* class method), 332
`replace_child()` (*DEVICE_OBJECT.VolTemplateProxy* class method), 435
`replace_child()` (*DRIVER_OBJECT.VolTemplateProxy* class method), 437
`replace_child()` (*elf.VolTemplateProxy* class method), 392
`replace_child()` (*elf_phdr.VolTemplateProxy* class method), 394
`replace_child()` (*elf_sym.VolTemplateProxy* class method), 396
`replace_child()` (*Enumeration.VolTemplateProxy* class method), 271
`replace_child()` (*EPROCESS.VolTemplateProxy* class method), 439
`replace_child()` (*ETHREAD.VolTemplateProxy* class method), 442
`replace_child()` (*EX_FAST_REF.VolTemplateProxy* class method), 444
`replace_child()` (*ExecutiveObject.VolTemplateProxy* class method), 490
`replace_child()` (*FILE_OBJECT.VolTemplateProxy* class method), 446
`replace_child()` (*fileglob.VolTemplateProxy* class method), 405
`replace_child()` (*files_struct.VolTemplateProxy* class method), 334
`replace_child()` (*Float.VolTemplateProxy* class method), 274
`replace_child()` (*fs_struct.VolTemplateProxy* class method), 336
`replace_child()` (*Function.VolTemplateProxy* class method), 276
`replace_child()` (*GenericIntelPro-*

cess.VolTemplateProxy class method), 320

`replace_child()` (*hist_entry.VolTemplateProxy class method*), 390

`replace_child()` (*HMAP_ENTRY.VolTemplateProxy class method*), 509

`replace_child()` (*ifnet.VolTemplateProxy class method*), 407

`replace_child()` (*IM-AGE_DOS_HEADER.VolTemplateProxy class method*), 485

`replace_child()` (*IM-AGE_NT_HEADERS.VolTemplateProxy class method*), 488

`replace_child()` (*inet_sock.VolTemplateProxy class method*), 338

`replace_child()` (*inpcb.VolTemplateProxy class method*), 409

`replace_child()` (*Integer.VolTemplateProxy class method*), 278

`replace_child()` (*kauth_scope.VolTemplateProxy class method*), 411

`replace_child()` (*KDDEBUG-GER_DATA64.VolTemplateProxy class method*), 473

`replace_child()` (*kernel_cap_struct.VolTemplateProxy class method*), 340

`replace_child()` (*kernel_cap_t.VolTemplateProxy class method*), 342

`replace_child()` (*KMUTANT.VolTemplateProxy class method*), 448

`replace_child()` (*kobject.VolTemplateProxy class method*), 345

`replace_child()` (*KSYSTEM_TIME.VolTemplateProxy class method*), 450

`replace_child()` (*KTHREAD.VolTemplateProxy class method*), 452

`replace_child()` (*LIST_ENTRY.VolTemplateProxy class method*), 454

`replace_child()` (*list_head.VolTemplateProxy class method*), 347

`replace_child()` (*maple_tree.VolTemplateProxy class method*), 349

`replace_child()` (*MFTAttribute.VolTemplateProxy class method*), 479

`replace_child()` (*MFTEntry.VolTemplateProxy class method*), 481

`replace_child()` (*MFTFileName.VolTemplateProxy class method*), 483

`replace_child()` (*mm_struct.VolTemplateProxy class method*), 351

`replace_child()` (*MMVAD.VolTemplateProxy class method*), 455

`replace_child()` (*MMVAD_SHORT.VolTemplateProxy class method*), 458

`replace_child()` (*mnt_namespace.VolTemplateProxy class method*), 353

`replace_child()` (*module.VolTemplateProxy class method*), 355

`replace_child()` (*mount.VolTemplateProxy class method*), 358

`replace_child()` (*net.VolTemplateProxy class method*), 361

`replace_child()` (*netlink_sock.VolTemplateProxy class method*), 363

`replace_child()` (*OBJECT_HEADER.VolTemplateProxy class method*), 492

`replace_child()` (*OBJECT_SYMBOLIC_LINK.VolTemplateProxy class method*), 461

`replace_child()` (*ObjectInterface.VolTemplateProxy class method*), 162

`replace_child()` (*ObjectTemplate method*), 299

`replace_child()` (*packet_sock.VolTemplateProxy class method*), 365

`replace_child()` (*PARTITION_ENTRY.VolTemplateProxy class method*), 475

`replace_child()` (*PARTITION_TABLE.VolTemplateProxy class method*), 478

`replace_child()` (*Pointer.VolTemplateProxy class method*), 281

`replace_child()` (*POOL_HEADER.VolTemplateProxy class method*), 494

`replace_child()` (*POOL_HEADER_VISTA.VolTemplateProxy class method*), 496

`replace_child()` (*POOL_TRACKER_BIG_PAGES.VolTemplateProxy class method*), 498

`replace_child()` (*PrimitiveObject.VolTemplateProxy class method*), 285

`replace_child()` (*proc.VolTemplateProxy class method*), 412

`replace_child()` (*qstr.VolTemplateProxy class method*), 367

`replace_child()` (*queue_entry.VolTemplateProxy class method*), 415

`replace_child()` (*ReferenceTemplate method*), 300

`replace_child()` (*SERVICE_HEADER.VolTemplateProxy class method*), 513

`replace_child()` (*SERVICE_RECORD.VolTemplateProxy class method*), 515

`replace_child()` (*SHARED_CACHE_MAP.VolTemplateProxy class method*), 463

`replace_child()` (*sock.VolTemplateProxy class method*), 458

- method*), 368
- `replace_child()` (*sockaddr.VolTemplateProxy class method*), 417
- `replace_child()` (*sockaddr_dl.VolTemplateProxy class method*), 418
- `replace_child()` (*socket.VolTemplateProxy class method*), 370, 420
- `replace_child()` (*String.VolTemplateProxy class method*), 286
- `replace_child()` (*struct_file.VolTemplateProxy class method*), 372
- `replace_child()` (*StructType.VolTemplateProxy class method*), 293
- `replace_child()` (*SUMMARY_DUMP.VolTemplateProxy class method*), 471
- `replace_child()` (*super_block.VolTemplateProxy class method*), 374
- `replace_child()` (*SymbolSpace.UnresolvedTemplate method*), 317
- `replace_child()` (*sysctl_oid.VolTemplateProxy class method*), 422
- `replace_child()` (*task_struct.VolTemplateProxy class method*), 376
- `replace_child()` (*Template method*), 164
- `replace_child()` (*TOKEN.VolTemplateProxy class method*), 465
- `replace_child()` (*UNICODE_STRING.VolTemplateProxy class method*), 467
- `replace_child()` (*UnionType.VolTemplateProxy class method*), 295
- `replace_child()` (*unix_sock.VolTemplateProxy class method*), 379
- `replace_child()` (*VACB.VolTemplateProxy class method*), 469
- `replace_child()` (*vfsmount.VolTemplateProxy class method*), 381
- `replace_child()` (*vm_area_struct.VolTemplateProxy class method*), 384
- `replace_child()` (*vm_map_entry.VolTemplateProxy class method*), 424
- `replace_child()` (*vm_map_object.VolTemplateProxy class method*), 426
- `replace_child()` (*vnnode.VolTemplateProxy class method*), 428
- `replace_child()` (*Void.VolTemplateProxy class method*), 296
- `replace_child()` (*vsock_sock.VolTemplateProxy class method*), 386
- `replace_child()` (*xdp_sock.VolTemplateProxy class method*), 388
- `replace_forward_references()` (*PdbReader method*), 519
- `replace_header_field()` (*IMAGE_DOS_HEADER method*), 487
- `require_interface_version()` (*in module volatility3.framework*), 69
- `RequirementInterface` (*class in volatility3.framework.interfaces.configuration*), 145
- `requirements` (*BooleanRequirement property*), 96
- `requirements` (*BytesRequirement property*), 97
- `requirements` (*ChoiceRequirement property*), 99
- `requirements` (*ClassRequirement property*), 138
- `requirements` (*ComplexListRequirement property*), 101
- `requirements` (*ConfigurableRequirementInterface property*), 141
- `requirements` (*ConstructableRequirementInterface property*), 143
- `requirements` (*IntRequirement property*), 102
- `requirements` (*LayerListRequirement property*), 104
- `requirements` (*ListRequirement property*), 105
- `requirements` (*ModuleRequirement property*), 107
- `requirements` (*MultiRequirement property*), 109
- `requirements` (*PluginRequirement property*), 110
- `requirements` (*RequirementInterface property*), 146
- `requirements` (*SimpleTypeRequirement property*), 147
- `requirements` (*StringRequirement property*), 112
- `requirements` (*SymbolTableRequirement property*), 114
- `requirements` (*TranslationLayerRequirement property*), 115
- `requirements` (*URIRequirement property*), 117
- `requirements` (*VersionRequirement property*), 118
- `reset()` (*PdbReader method*), 519
- `ResourceAccessor` (*class in volatility3.framework.layers.resources*), 238
- `retrieve_pdb()` (*PdbRetriever method*), 519
- `rfind()` (*Bytes method*), 263
- `rfind()` (*HexBytes method*), 311
- `rfind()` (*MultiTypeData method*), 315
- `rfind()` (*String method*), 290
- `rindex()` (*Bytes method*), 263
- `rindex()` (*HexBytes method*), 311
- `rindex()` (*MultiTypeData method*), 315
- `rindex()` (*String method*), 291
- `rjust()` (*Bytes method*), 263
- `rjust()` (*HexBytes method*), 311
- `rjust()` (*MultiTypeData method*), 315
- `rjust()` (*String method*), 291
- `root_cell_offset` (*RegistryHive property*), 236
- `round()` (*in module volatility3.framework.renderers.conversion*), 304
- `row_count` (*TreeGrid property*), 303
- `RowStructureConstructor()` (*in module volatility3.framework.renderers*), 302
- `rpartition()` (*Bytes method*), 263
- `rpartition()` (*HexBytes method*), 311

`rpartition()` (*MultiTypeData method*), 315
`rpartition()` (*String method*), 291
`rsplit()` (*Bytes method*), 263
`rsplit()` (*HexBytes method*), 311
`rsplit()` (*MultiTypeData method*), 315
`rsplit()` (*String method*), 291
`rstrip()` (*Bytes method*), 264
`rstrip()` (*HexBytes method*), 311
`rstrip()` (*MultiTypeData method*), 315
`rstrip()` (*String method*), 291
`run()` (*ABCKmsg method*), 579
`run()` (*ADS method*), 702
`run()` (*Banners method*), 754
`run()` (*Bash method*), 560, 609
`run()` (*BigPools method*), 656
`run()` (*Cachedump method*), 658
`run()` (*Callbacks method*), 661
`run()` (*Capabilities method*), 562
`run()` (*Check_afinfo method*), 564
`run()` (*Check_creds method*), 565
`run()` (*Check_idt method*), 567
`run()` (*Check_modules method*), 569
`run()` (*Check_syscall method*), 570, 611
`run()` (*Check_sysctl method*), 612
`run()` (*Check_trap_table method*), 614
`run()` (*CmdLine method*), 663
`run()` (*CommandLine method*), 49
`run()` (*ConfigWriter method*), 756
`run()` (*Crashinfo method*), 665
`run()` (*DeviceTree method*), 666
`run()` (*DllList method*), 669
`run()` (*DriverIrp method*), 670
`run()` (*DriverModule method*), 672
`run()` (*DriverScan method*), 674
`run()` (*DumpFiles method*), 676
`run()` (*Elfs method*), 572
`run()` (*Envvars method*), 574, 678
`run()` (*FileScan method*), 679
`run()` (*FrameworkInfo method*), 757
`run()` (*GetServiceSIDs method*), 681
`run()` (*GetSIDs method*), 683
`run()` (*Handles method*), 685
`run()` (*Hashdump method*), 688
`run()` (*HiveList method*), 649
`run()` (*HiveScan method*), 650
`run()` (*Ifconfig method*), 616
`run()` (*in module volatility3.framework.automagic*), 69
`run()` (*Info method*), 690
`run()` (*IOMem method*), 576
`run()` (*IsfInfo method*), 759
`run()` (*JobLinks method*), 692
`run()` (*Kauth_listeners method*), 617
`run()` (*Kauth_scopes method*), 619
`run()` (*Kevents method*), 621
`run()` (*Keyboard_notifiers method*), 577
`run()` (*Kmsg method*), 581
`run()` (*KmsgFiveTen method*), 582
`run()` (*KmsgLegacy method*), 584
`run()` (*LayerWriter method*), 761
`run()` (*LdrModules method*), 694
`run()` (*List_Files method*), 623
`run()` (*Lsadump method*), 695
`run()` (*Lsmode method*), 586, 624
`run()` (*Lsof method*), 588, 626
`run()` (*Malfind method*), 589, 628, 698
`run()` (*Maps method*), 593, 633
`run()` (*MBRScan method*), 699
`run()` (*Memmap method*), 701
`run()` (*MFTScan method*), 704
`run()` (*ModScan method*), 706
`run()` (*Modules method*), 709
`run()` (*Mount method*), 629
`run()` (*MountInfo method*), 591
`run()` (*MutantScan method*), 711
`run()` (*NetScan method*), 713
`run()` (*NetStat method*), 718
`run()` (*Netstat method*), 631
`run()` (*PluginInterface method*), 168
`run()` (*PoolScanner method*), 722
`run()` (*PrintKey method*), 653
`run()` (*Privs method*), 725
`run()` (*PsAux method*), 595
`run()` (*Psaux method*), 634
`run()` (*PsList method*), 598, 638, 728
`run()` (*PsScan method*), 600, 730
`run()` (*PsTree method*), 602, 639, 732
`run()` (*Sessions method*), 734
`run()` (*Skeleton_Key_Check method*), 735
`run()` (*Socket_filters method*), 641
`run()` (*Sockstat method*), 604
`run()` (*SSDT method*), 737
`run()` (*Strings method*), 739
`run()` (*SvcScan method*), 741
`run()` (*SymlinkScan method*), 743
`run()` (*Timeliner method*), 765
`run()` (*Timers method*), 643
`run()` (*Trustedbsd method*), 644
`run()` (*tty_check method*), 605
`run()` (*UserAssist method*), 655
`run()` (*VadInfo method*), 745
`run()` (*VadWalk method*), 747
`run()` (*VadYaraScan method*), 749
`run()` (*VerInfo method*), 751
`run()` (*VFSevents method*), 646
`run()` (*VirtMap method*), 753
`run()` (*VmaYaraScan method*), 607
`run()` (*VolShell method*), 50
`run()` (*Volshell method*), 54, 57, 60, 63

run() (*YaraScan* method), 766
 run_all() (*ABCKmsg* class method), 579
 run_all() (*KmsgFiveTen* class method), 583
 run_all() (*KmsgLegacy* class method), 584
 run_script() (*Volshell* method), 55, 57, 60, 63

S

sanitize_filename() (*FileHandlerInterface* static method), 166
 sanitize_filename() (*NullFileHandler* static method), 52
 sanitize_name() (*TreeGrid* static method), 170, 303
 save_vacb() (*SHARED_CACHE_MAP* method), 464
 SB_DIRSYNC (*super_block* attribute), 374
 SB_I_VERSION (*super_block* attribute), 374
 SB_KERNMOUNT (*super_block* attribute), 374
 SB_LAZYTIME (*super_block* attribute), 374
 SB_MANDLOCK (*super_block* attribute), 374
 SB_NOATIME (*super_block* attribute), 374
 SB_NODEV (*super_block* attribute), 374
 SB_NODIRATIME (*super_block* attribute), 374
 SB_NOEXEC (*super_block* attribute), 374
 SB_NOSUID (*super_block* attribute), 374
 sb_opts (*MountInfoData* attribute), 592
 SB_OPTS (*super_block* attribute), 374
 SB_POSIXACL (*super_block* attribute), 374
 SB_RDONLY (*super_block* attribute), 374
 SB_SILENT (*super_block* attribute), 374
 SB_SYNCHRONOUS (*super_block* attribute), 374
 scan() (*AVMLLayer* method), 183
 scan() (*BufferDataLayer* method), 227
 scan() (*DataLayerInterface* method), 156
 scan() (*Elf64Layer* method), 193
 scan() (*FileLayer* method), 229
 scan() (*Intel* method), 196
 scan() (*Intel32e* method), 199
 scan() (*IntelPAE* method), 202
 scan() (*LimeLayer* method), 216
 scan() (*LinearlyMappedLayer* method), 219
 scan() (*NetScan* class method), 714
 scan() (*NonLinearlySegmentedLayer* method), 240
 scan() (*PdbMSFStream* method), 221
 scan() (*PdbMultiStreamFormat* method), 224
 scan() (*QemuSuspendLayer* method), 232
 scan() (*RegistryHive* method), 236
 scan() (*SegmentedLayer* method), 242
 scan() (*TranslationLayerInterface* method), 160
 scan() (*VmwareLayer* method), 245
 scan() (*WindowsCrashDump32Layer* method), 187
 scan() (*WindowsCrashDump64Layer* method), 190
 scan() (*WindowsIntel* method), 205
 scan() (*WindowsIntel32e* method), 207
 scan() (*WindowsIntelPAE* method), 210
 scan() (*WindowsMixin* method), 213
 scan() (*XenCoreDumpLayer* method), 248
 scan_drivers() (*DriverScan* class method), 674
 scan_files() (*FileScan* class method), 679
 scan_hives() (*HiveScan* class method), 651
 scan_modules() (*ModScan* class method), 706
 scan_mutants() (*MutantScan* class method), 711
 scan_processes() (*PsScan* class method), 730
 scan_symlinks() (*SymlinkScan* class method), 743
 scan_tasks() (*PsScan* class method), 600
 scannable_sections() (*VirtMap* class method), 753
 ScannerInterface (class in *volatility3.framework.interfaces.layers*), 158
 search() (*MultiRegexp* method), 181
 search() (*MultiStringScanner* method), 180
 section_strtab (module property), 357
 section_syntab (module property), 357
 seek() (*FileHandlerInterface* method), 166
 seek() (*NullFileHandler* method), 52
 seekable() (*FileHandlerInterface* method), 166
 seekable() (*NullFileHandler* method), 52
 SEGMENT_FLAG_COMPRESS (*QemuSuspendLayer* attribute), 231
 SEGMENT_FLAG_CONTINUE (*QemuSuspendLayer* attribute), 231
 SEGMENT_FLAG_EOS (*QemuSuspendLayer* attribute), 231
 SEGMENT_FLAG_HOOK (*QemuSuspendLayer* attribute), 231
 SEGMENT_FLAG_MEM_SIZE (*QemuSuspendLayer* attribute), 231
 SEGMENT_FLAG_PAGE (*QemuSuspendLayer* attribute), 231
 SEGMENT_FLAG_XBZRLE (*QemuSuspendLayer* attribute), 231
 SegmentedLayer (class in *volatility3.framework.layers.segmented*), 241
 separator (*HierarchicalDict* property), 144
 separator (*WindowsIdentifier* attribute), 90
 SERVICE_HEADER (class in *volatility3.framework.symbols.windows.extensions.services*), 512
 SERVICE_HEADER.VolTemplateProxy (class in *volatility3.framework.symbols.windows.extensions.services*), 513
 SERVICE_RECORD (class in *volatility3.framework.symbols.windows.extensions.services*), 514
 SERVICE_RECORD.VolTemplateProxy (class in *volatility3.framework.symbols.windows.extensions.services*), 514
 Sessions (class in *volatility3.plugins.windows.sessions*), 733
 set_defaults() (*HelpfulArgParser* method), 68
 set_kernel_virtual_offset() (*KernelPDBScanner* method), 81

`set_open_method()` (*ADS method*), 703
`set_open_method()` (*Banners method*), 755
`set_open_method()` (*Bash method*), 560, 609
`set_open_method()` (*BigPools method*), 657
`set_open_method()` (*Cachedump method*), 658
`set_open_method()` (*Callbacks method*), 662
`set_open_method()` (*Capabilities method*), 562
`set_open_method()` (*Check_afinfo method*), 564
`set_open_method()` (*Check_creds method*), 566
`set_open_method()` (*Check_idt method*), 567
`set_open_method()` (*Check_modules method*), 569
`set_open_method()` (*Check_syscall method*), 571, 611
`set_open_method()` (*Check_sysctl method*), 613
`set_open_method()` (*Check_trap_table method*), 614
`set_open_method()` (*CmdLine method*), 664
`set_open_method()` (*ConfigWriter method*), 756
`set_open_method()` (*Crashinfo method*), 665
`set_open_method()` (*DeviceTree method*), 667
`set_open_method()` (*DllList method*), 669
`set_open_method()` (*DriverIrp method*), 670
`set_open_method()` (*DriverModule method*), 672
`set_open_method()` (*DriverScan method*), 674
`set_open_method()` (*DumpFiles method*), 676
`set_open_method()` (*Elfs method*), 572
`set_open_method()` (*Envvars method*), 574, 678
`set_open_method()` (*FileScan method*), 680
`set_open_method()` (*FrameworkInfo method*), 758
`set_open_method()` (*GetServiceSIDs method*), 681
`set_open_method()` (*GetSIDs method*), 683
`set_open_method()` (*Handles method*), 685
`set_open_method()` (*Hashdump method*), 688
`set_open_method()` (*HiveList method*), 649
`set_open_method()` (*HiveScan method*), 651
`set_open_method()` (*Ifconfig method*), 616
`set_open_method()` (*Info method*), 691
`set_open_method()` (*IOMem method*), 576
`set_open_method()` (*IsfInfo method*), 759
`set_open_method()` (*JobLinks method*), 692
`set_open_method()` (*Kauth_listeners method*), 617
`set_open_method()` (*Kauth_scopes method*), 619
`set_open_method()` (*Kevents method*), 621
`set_open_method()` (*Keyboard_notifiers method*), 578
`set_open_method()` (*Kmsg method*), 581
`set_open_method()` (*LayerWriter method*), 761
`set_open_method()` (*LdrModules method*), 694
`set_open_method()` (*List_Files method*), 623
`set_open_method()` (*Lsadump method*), 696
`set_open_method()` (*Lsmodule method*), 586, 625
`set_open_method()` (*Lsof method*), 588, 626
`set_open_method()` (*Malfind method*), 589, 628, 698
`set_open_method()` (*Maps method*), 594, 633
`set_open_method()` (*MBRScan method*), 700
`set_open_method()` (*Memmap method*), 701
`set_open_method()` (*MFTScan method*), 704
`set_open_method()` (*ModScan method*), 707
`set_open_method()` (*Modules method*), 709
`set_open_method()` (*Mount method*), 630
`set_open_method()` (*MountInfo method*), 591
`set_open_method()` (*MutantScan method*), 711
`set_open_method()` (*NetScan method*), 714
`set_open_method()` (*NetStat method*), 719
`set_open_method()` (*Netstat method*), 631
`set_open_method()` (*PluginInterface method*), 168
`set_open_method()` (*PoolScanner method*), 722
`set_open_method()` (*PrintKey method*), 653
`set_open_method()` (*Privs method*), 725
`set_open_method()` (*PsAux method*), 596
`set_open_method()` (*Psaux method*), 634
`set_open_method()` (*PsList method*), 598, 638, 728
`set_open_method()` (*PsScan method*), 600, 730
`set_open_method()` (*PsTree method*), 602, 640, 732
`set_open_method()` (*Sessions method*), 734
`set_open_method()` (*Skeleton_Key_Check method*), 735
`set_open_method()` (*Socket_filters method*), 641
`set_open_method()` (*Sockstat method*), 604
`set_open_method()` (*SSDT method*), 737
`set_open_method()` (*Strings method*), 739
`set_open_method()` (*SvcScan method*), 741
`set_open_method()` (*SymlinkScan method*), 743
`set_open_method()` (*Timeliner method*), 765
`set_open_method()` (*Timers method*), 643
`set_open_method()` (*Trustedbsd method*), 644
`set_open_method()` (*tty_check method*), 606
`set_open_method()` (*UserAssist method*), 655
`set_open_method()` (*VadInfo method*), 745
`set_open_method()` (*VadWalk method*), 747
`set_open_method()` (*VadYaraScan method*), 749
`set_open_method()` (*VerInfo method*), 751
`set_open_method()` (*VFSevents method*), 646
`set_open_method()` (*VirtMap method*), 753
`set_open_method()` (*VmaYaraScan method*), 607
`set_open_method()` (*Volshell method*), 55, 57, 60, 63
`set_open_method()` (*YaraScan method*), 766
`set_type_class()` (*BaseSymbolTableInterface method*), 173
`set_type_class()` (*BashIntermedSymbols method*), 400
`set_type_class()` (*IntermediateSymbolTable method*), 529
`set_type_class()` (*ISFormatTable method*), 525
`set_type_class()` (*LinuxKernelIntermedSymbols method*), 324
`set_type_class()` (*MacKernelIntermedSymbols method*), 403
`set_type_class()` (*NativeTable method*), 555
`set_type_class()` (*NativeTableInterface method*), 175

- `set_type_class()` (*SymbolTableInterface* method), 179
- `set_type_class()` (*Version1Format* method), 531
- `set_type_class()` (*Version2Format* method), 534
- `set_type_class()` (*Version3Format* method), 537
- `set_type_class()` (*Version4Format* method), 540
- `set_type_class()` (*Version5Format* method), 543
- `set_type_class()` (*Version6Format* method), 546
- `set_type_class()` (*Version7Format* method), 549
- `set_type_class()` (*Version8Format* method), 552
- `set_type_class()` (*WindowsKernellIntermedSymbols* method), 432
- `setter()` (*classproperty* method), 47
- `setup_logging()` (*CommandLine* class method), 49
- `setup_logging()` (*VolShell* class method), 50
- `SHARED_CACHE_MAP` (class in *volatility3.framework.symbols.windows.extensions*), 462
- `SHARED_CACHE_MAP.VolTemplateProxy` (class in *volatility3.framework.symbols.windows.extensions*), 462
- `sid_to_key()` (*Hashdump* class method), 688
- `sidbytes_to_key()` (*Hashdump* class method), 688
- `SIGNATURE` (*WindowsCrashDump32Layer* attribute), 185
- `SIGNATURE` (*WindowsCrashDump64Layer* attribute), 188
- `signed` (*DataFormatInfo* attribute), 270
- `SimpleTypeRequirement` (class in *volatility3.framework.interfaces.configuration*), 146
- `size` (*ObjectTemplate* property), 299
- `size` (*ReferenceTemplate* property), 300
- `size` (*SizedModule* property), 133
- `size` (*SymbolSpace.UnresolvedTemplate* property), 317
- `size` (*Template* property), 164
- `size()` (*AggregateType.VolTemplateProxy* class method), 250
- `size()` (*Array.VolTemplateProxy* class method), 252
- `size()` (*BitField.VolTemplateProxy* class method), 253
- `size()` (*Boolean.VolTemplateProxy* class method), 257
- `size()` (*bpf_prog.VolTemplateProxy* class method), 327
- `size()` (*bt_sock.VolTemplateProxy* class method), 328
- `size()` (*Bytes.VolTemplateProxy* class method), 260
- `size()` (*Char.VolTemplateProxy* class method), 265
- `size()` (*ClassType.VolTemplateProxy* class method), 268
- `size()` (*CM_KEY_BODY.VolTemplateProxy* class method), 503
- `size()` (*CM_KEY_NODE.VolTemplateProxy* class method), 505
- `size()` (*CM_KEY_VALUE.VolTemplateProxy* class method), 507
- `size()` (*CMHIVE.VolTemplateProxy* class method), 501
- `size()` (*CONTROL_AREA.VolTemplateProxy* class method), 433
- `size()` (*cred.VolTemplateProxy* class method), 330
- `size()` (*dentry.VolTemplateProxy* class method), 332
- `size()` (*DEVICE_OBJECT.VolTemplateProxy* class method), 435
- `size()` (*DRIVER_OBJECT.VolTemplateProxy* class method), 437
- `size()` (*elf.VolTemplateProxy* class method), 392
- `size()` (*elf_phdr.VolTemplateProxy* class method), 394
- `size()` (*elf_sym.VolTemplateProxy* class method), 396
- `size()` (*Enumeration.VolTemplateProxy* class method), 271
- `size()` (*EPROCESS.VolTemplateProxy* class method), 439
- `size()` (*ETHREAD.VolTemplateProxy* class method), 442
- `size()` (*EX_FAST_REF.VolTemplateProxy* class method), 444
- `size()` (*ExecutiveObject.VolTemplateProxy* class method), 490
- `size()` (*FILE_OBJECT.VolTemplateProxy* class method), 446
- `size()` (*fileglob.VolTemplateProxy* class method), 405
- `size()` (*files_struct.VolTemplateProxy* class method), 334
- `size()` (*Float.VolTemplateProxy* class method), 274
- `size()` (*fs_struct.VolTemplateProxy* class method), 336
- `size()` (*Function.VolTemplateProxy* class method), 277
- `size()` (*GenericIntelProcess.VolTemplateProxy* class method), 320
- `size()` (*hist_entry.VolTemplateProxy* class method), 390
- `size()` (*HMAP_ENTRY.VolTemplateProxy* class method), 509
- `size()` (*ifnet.VolTemplateProxy* class method), 407
- `size()` (*IMAGE_DOS_HEADER.VolTemplateProxy* class method), 485
- `size()` (*IMAGE_NT_HEADERS.VolTemplateProxy* class method), 488
- `size()` (*inet_sock.VolTemplateProxy* class method), 338
- `size()` (*inpcb.VolTemplateProxy* class method), 409
- `size()` (*Integer.VolTemplateProxy* class method), 278
- `size()` (*kauth_scope.VolTemplateProxy* class method), 411
- `size()` (*KDDEBUGGER_DATA64.VolTemplateProxy* class method), 473
- `size()` (*kernel_cap_struct.VolTemplateProxy* class method), 340
- `size()` (*kernel_cap_t.VolTemplateProxy* class method), 343
- `size()` (*KMUTANT.VolTemplateProxy* class method), 448
- `size()` (*kobject.VolTemplateProxy* class method), 345
- `size()` (*KSYSTEM_TIME.VolTemplateProxy* class method), 450
- `size()` (*KTHREAD.VolTemplateProxy* class method),

452
size() (*LIST_ENTRY.VolTemplateProxy* class method), 454
size() (*list_head.VolTemplateProxy* class method), 347
size() (*maple_tree.VolTemplateProxy* class method), 349
size() (*MFTAttribute.VolTemplateProxy* class method), 480
size() (*MFTEntry.VolTemplateProxy* class method), 481
size() (*MFTFileName.VolTemplateProxy* class method), 483
size() (*mm_struct.VolTemplateProxy* class method), 351
size() (*MMVAD.VolTemplateProxy* class method), 456
size() (*MMVAD_SHORT.VolTemplateProxy* class method), 458
size() (*mnt_namespace.VolTemplateProxy* class method), 353
size() (*module.VolTemplateProxy* class method), 355
size() (*mount.VolTemplateProxy* class method), 358
size() (*net.VolTemplateProxy* class method), 361
size() (*netlink_sock.VolTemplateProxy* class method), 363
size() (*OBJECT_HEADER.VolTemplateProxy* class method), 492
size() (*OBJECT_SYMBOLIC_LINK.VolTemplateProxy* class method), 461
size() (*ObjectInterface.VolTemplateProxy* class method), 162
size() (*packet_sock.VolTemplateProxy* class method), 365
size() (*PARTITION_ENTRY.VolTemplateProxy* class method), 475
size() (*PARTITION_TABLE.VolTemplateProxy* class method), 478
size() (*Pointer.VolTemplateProxy* class method), 281
size() (*POOL_HEADER.VolTemplateProxy* class method), 494
size() (*POOL_HEADER_VISTA.VolTemplateProxy* class method), 496
size() (*POOL_TRACKER_BIG_PAGES.VolTemplateProxy* class method), 498
size() (*PrimitiveObject.VolTemplateProxy* class method), 285
size() (*proc.VolTemplateProxy* class method), 413
size() (*qstr.VolTemplateProxy* class method), 367
size() (*queue_entry.VolTemplateProxy* class method), 415
size() (*SERVICE_HEADER.VolTemplateProxy* class method), 513
size() (*SERVICE_RECORD.VolTemplateProxy* class method), 515
size() (*SHARED_CACHE_MAP.VolTemplateProxy* class method), 463
size() (*sock.VolTemplateProxy* class method), 368
size() (*sockaddr.VolTemplateProxy* class method), 417
size() (*sockaddr_dl.VolTemplateProxy* class method), 419
size() (*socket.VolTemplateProxy* class method), 370, 420
size() (*String.VolTemplateProxy* class method), 287
size() (*struct_file.VolTemplateProxy* class method), 372
size() (*StructType.VolTemplateProxy* class method), 293
size() (*SUMMARY_DUMP.VolTemplateProxy* class method), 471
size() (*super_block.VolTemplateProxy* class method), 375
size() (*sysctl_oid.VolTemplateProxy* class method), 422
size() (*task_struct.VolTemplateProxy* class method), 377
size() (*TOKEN.VolTemplateProxy* class method), 465
size() (*UNICODE_STRING.VolTemplateProxy* class method), 467
size() (*UnionType.VolTemplateProxy* class method), 295
size() (*unix_sock.VolTemplateProxy* class method), 379
size() (*VACB.VolTemplateProxy* class method), 469
size() (*vfsmount.VolTemplateProxy* class method), 381
size() (*vm_area_struct.VolTemplateProxy* class method), 384
size() (*vm_map_entry.VolTemplateProxy* class method), 424
size() (*vm_map_object.VolTemplateProxy* class method), 426
size() (*vnode.VolTemplateProxy* class method), 428
size() (*Void.VolTemplateProxy* class method), 296
size() (*vsock_sock.VolTemplateProxy* class method), 386
size() (*xdp_sock.VolTemplateProxy* class method), 388
SizedModule (class in *volatility3.framework.contexts*), 131
Skeleton_Key_Check (class in *volatility3.plugins.windows.skeleton_key_check*), 734
SnappyException, 184
sock (class in *volatility3.framework.symbols.linux.extensions*), 368
sock.VolTemplateProxy (class in *volatility3.framework.symbols.linux.extensions*), 368
sockaddr (class in *volatility3.framework.symbols.mac.extensions*), 416
sockaddr.VolTemplateProxy (class in *volatility3.framework.symbols.mac.extensions*), 416
sockaddr_dl (class in *volatility3.framework.symbols.mac.extensions*),

- 418
- `sockaddr_dl()` (*ifnet method*), 408
- `sockaddr_dl.VolTemplateProxy` (class in *volatility3.framework.symbols.mac.extensions*), 418
- `socket` (class in *volatility3.framework.symbols.linux.extensions*), 370
- `socket` (class in *volatility3.framework.symbols.mac.extensions*), 420
- `socket.VolTemplateProxy` (class in *volatility3.framework.symbols.linux.extensions*), 370
- `socket.VolTemplateProxy` (class in *volatility3.framework.symbols.mac.extensions*), 420
- `Socket_filters` (class in *volatility3.plugins.mac.socket_filters*), 640
- `SockHandlers` (class in *volatility3.plugins.linux.sockstat*), 602
- `Sockstat` (class in *volatility3.plugins.linux.sockstat*), 603
- `splice()` (*HierarchicalDict method*), 144
- `split()` (*Bytes method*), 264
- `split()` (*HexBytes method*), 311
- `split()` (*MultiTypeData method*), 315
- `split()` (*String method*), 291
- `splitlines()` (*Bytes method*), 264
- `splitlines()` (*HexBytes method*), 311
- `splitlines()` (*MultiTypeData method*), 315
- `splitlines()` (*String method*), 291
- `SQLITE_CACHE_PERIOD` (in module *volatility3.framework.constants*), 122
- `SqliteCache` (class in *volatility3.framework.automagic.symbol_cache*), 87
- `SSDT` (class in *volatility3.plugins.windows.ssd*), 736
- `st_dev` (*MountInfoData attribute*), 592
- `stack()` (*AVMLStacker class method*), 184
- `stack()` (*Elf64Stacker class method*), 194
- `stack()` (*LayerStacker method*), 84
- `stack()` (*LimeStacker class method*), 216
- `stack()` (*LinuxIntelStacker class method*), 72
- `stack()` (*MacIntelStacker class method*), 74
- `stack()` (*QemuStacker class method*), 230
- `stack()` (*StackerLayerInterface class method*), 136
- `stack()` (*VmwareStacker class method*), 246
- `stack()` (*WindowsCrashDumpStacker class method*), 191
- `stack()` (*WindowsIntelStacker class method*), 95
- `stack()` (*XenCoreDumpStacker class method*), 249
- `stack_layer()` (*LayerStacker class method*), 84
- `stack_order` (*AVMLStacker attribute*), 184
- `stack_order` (*Elf64Stacker attribute*), 194
- `stack_order` (*LimeStacker attribute*), 217
- `stack_order` (*LinuxIntelStacker attribute*), 72
- `stack_order` (*MacIntelStacker attribute*), 74
- `stack_order` (*QemuStacker attribute*), 230
- `stack_order` (*StackerLayerInterface attribute*), 137
- `stack_order` (*VmwareStacker attribute*), 246
- `stack_order` (*WindowsCrashDumpStacker attribute*), 191
- `stack_order` (*WindowsIntelStacker attribute*), 95
- `stack_order` (*XenCoreDumpStacker attribute*), 249
- `stacker_slow_warning()` (*AVMLStacker class method*), 184
- `stacker_slow_warning()` (*Elf64Stacker class method*), 194
- `stacker_slow_warning()` (*LimeStacker class method*), 217
- `stacker_slow_warning()` (*LinuxIntelStacker class method*), 72
- `stacker_slow_warning()` (*MacIntelStacker class method*), 74
- `stacker_slow_warning()` (*QemuStacker class method*), 230
- `stacker_slow_warning()` (*StackerLayerInterface class method*), 137
- `stacker_slow_warning()` (*VmwareStacker class method*), 246
- `stacker_slow_warning()` (*WindowsCrashDumpStacker class method*), 191
- `stacker_slow_warning()` (*WindowsIntelStacker class method*), 95
- `stacker_slow_warning()` (*XenCoreDumpStacker class method*), 249
- `StackerLayerInterface` (class in *volatility3.framework.interfaces.automagic*), 136
- `startswith()` (*Bytes method*), 264
- `startswith()` (*HexBytes method*), 312
- `startswith()` (*MultiTypeData method*), 316
- `startswith()` (*String method*), 291
- `String` (class in *volatility3.framework.objects*), 286
- `String` (*UNICODE_STRING property*), 467
- `String.VolTemplateProxy` (class in *volatility3.framework.objects*), 286
- `StringRequirement` (class in *volatility3.framework.configuration.requirements*), 111
- `Strings` (class in *volatility3.plugins.windows.strings*), 738
- `strings_pattern` (*Strings attribute*), 739
- `strip()` (*Bytes method*), 264
- `strip()` (*HexBytes method*), 312
- `strip()` (*MultiTypeData method*), 316
- `strip()` (*String method*), 292
- `struct_file` (class in *volatility3.framework.symbols.linux.extensions*),

- 371
- `struct_file.VolTemplateProxy` (class in `volatility3.framework.symbols.linux.extensions`), 372
- `StructType` (class in `volatility3.framework.objects`), 292
- `StructType.VolTemplateProxy` (class in `volatility3.framework.objects`), 292
- `structure` (Intel attribute), 197
- `structure` (Intel32e attribute), 200
- `structure` (IntelPAE attribute), 202
- `structure` (WindowsIntel attribute), 205
- `structure` (WindowsIntel32e attribute), 208
- `structure` (WindowsIntelPAE attribute), 210
- `structure` (WindowsMixin attribute), 213
- `structured_output` (CLIRenderer attribute), 64
- `structured_output` (CSVRenderer attribute), 64
- `structured_output` (JsonLinesRenderer attribute), 65
- `structured_output` (JsonRenderer attribute), 65
- `structured_output` (NoneRenderer attribute), 65
- `structured_output` (PrettyTextRenderer attribute), 66
- `structured_output` (QuickTextRenderer attribute), 66
- `SUMMARY_DUMP` (class in `volatility3.framework.symbols.windows.extensions.crash_symbols`), 470
- `SUMMARY_DUMP.VolTemplateProxy` (class in `volatility3.framework.symbols.windows.extensions.crash_symbols`), 471
- `super_block` (class in `volatility3.framework.symbols.linux.extensions`), 373
- `super_block.VolTemplateProxy` (class in `volatility3.framework.symbols.linux.extensions`), 374
- `supported_dumptypes` (WindowsCrashDump32Layer attribute), 187
- `supported_dumptypes` (WindowsCrashDump64Layer attribute), 190
- `SvcScan` (class in `volatility3.plugins.windows.svcscan`), 740
- `swapcase()` (Bytes method), 264
- `swapcase()` (HexBytes method), 312
- `swapcase()` (MultiTypeData method), 316
- `swapcase()` (String method), 292
- `SwappedInvalidAddressException`, 557
- `SYMBOL` (SymbolType attribute), 319
- `SYMBOL_BASEPATHS` (in module `volatility3.framework.constants`), 122
- `symbol_class` (LinuxSymbolFinder attribute), 74
- `symbol_class` (MacSymbolFinder attribute), 76
- `symbol_class` (SymbolFinder attribute), 92
- `symbol_space` (Context property), 127
- `symbol_space` (ContextInterface property), 150
- `symbol_table_from_offset()` (PDBUtility class method), 521
- `symbol_table_from_pdb()` (PDBUtility class method), 521
- `symbol_table_is_64bit()` (in module `volatility3.framework.symbols`), 319
- `symbol_table_name` (ConfigurableModule property), 125
- `symbol_table_name` (Module property), 129
- `symbol_table_name` (ModuleInterface property), 153
- `symbol_table_name` (SizedModule property), 133
- `SymbolCacheMagic` (class in `volatility3.framework.automagic.symbol_cache`), 88
- `SymbolError`, 557
- `SymbolFinder` (class in `volatility3.framework.automagic.symbol_finder`), 90
- `SymbolInterface` (class in `volatility3.framework.interfaces.symbols`), 175
- `symbols` (BaseSymbolTableInterface property), 173
- `symbols` (BashIntermedSymbols property), 400
- `symbols` (ConfigurableModule property), 125
- `symbols` (IntermediateSymbolTable property), 529
- `symbols` (ISFormatTable property), 525
- `symbols` (LinuxKernelIntermedSymbols property), 324
- `symbols` (MacKernelIntermedSymbols property), 403
- `symbols` (Module property), 130
- `symbols` (NativeTable property), 555
- `symbols` (NativeTableInterface property), 175
- `symbols` (SizedModule property), 133
- `symbols` (SymbolTableInterface property), 180
- `symbols` (Version1Format property), 532
- `symbols` (Version2Format property), 534
- `symbols` (Version3Format property), 537
- `symbols` (Version4Format property), 540
- `symbols` (Version5Format property), 543
- `symbols` (Version6Format property), 546
- `symbols` (Version7Format property), 549
- `symbols` (Version8Format property), 552
- `symbols` (WindowsKernelIntermedSymbols property), 432
- `symbols()` (ModuleInterface method), 153
- `SymbolSpace` (class in `volatility3.framework.symbols`), 316
- `SymbolSpace.UnresolvedTemplate` (class in `volatility3.framework.symbols`), 316
- `SymbolSpaceError`, 557
- `SymbolSpaceInterface` (class in `volatility3.framework.interfaces.symbols`), 176
- `SymbolTableInterface` (class in `volatility3.framework.interfaces.symbols`), 177
- `SymbolTableRequirement` (class in `volatility3.framework.configuration.requirements`), 112
- `SymbolType` (class in `volatility3.framework.symbols`), 319

- SymlinkScan (class in volatility3.plugins.windows.symlinkscan), 742
 symtab_checks() (ABCKmsg class method), 579
 symtab_checks() (KmsgFiveTen class method), 583
 symtab_checks() (KmsgLegacy class method), 584
 sysctl_oid (class in volatility3.framework.symbols.mac.extensions), 421
 sysctl_oid.VolTemplateProxy (class in volatility3.framework.symbols.mac.extensions), 422
- ## T
- tab_stop() (PrettyTextRenderer method), 66
 TASK_RUNNING (DescExitStateEnum attribute), 599
 task_struct (class in volatility3.framework.symbols.linux.extensions), 376
 task_struct.VolTemplateProxy (class in volatility3.framework.symbols.linux.extensions), 376
 TaskData (class in volatility3.plugins.linux.capabilities), 563
 tell() (FileHandlerInterface method), 166
 tell() (NullFileHandler method), 52
 Template (class in volatility3.framework.interfaces.objects), 163
 test_sets (WindowsIntelStacker attribute), 95
 tests (PageMapScanner attribute), 93
 tgid (TaskData attribute), 563
 thread_safe (BytesScanner attribute), 180
 thread_safe (MultiStringScanner attribute), 181
 thread_safe (PageMapScanner attribute), 93
 thread_safe (PdbSignatureScanner attribute), 522
 thread_safe (PoolHeaderScanner attribute), 719
 thread_safe (RegExScanner attribute), 181
 thread_safe (ScannerInterface attribute), 158
 thread_safe (YaraScanner attribute), 767
 Threading (Parallelism attribute), 120
 Timeliner (class in volatility3.plugins.timeliner), 764
 TimeLinerInterface (class in volatility3.plugins.timeliner), 762
 TimeLinerType (class in volatility3.plugins.timeliner), 762
 timer_filters (Kevents attribute), 621
 Timers (class in volatility3.plugins.mac.timers), 642
 title() (Bytes method), 264
 title() (HexBytes method), 312
 title() (MultiTypeData method), 316
 title() (String method), 292
 to_bytes() (Bin method), 306
 to_bytes() (BitField method), 255
 to_bytes() (Boolean method), 259
 to_bytes() (Char method), 267
 to_bytes() (Enumeration method), 273
 to_bytes() (Hex method), 308
 to_bytes() (Integer method), 280
 to_bytes() (Parallelism method), 121
 to_bytes() (Pointer method), 284
 to_bytes() (PoolType method), 724
 to_bytes() (RegKeyFlags method), 511
 to_bytes() (TimeLinerType method), 763
 to_list() (LIST_ENTRY method), 455
 to_list() (list_head method), 348
 TOKEN (class in volatility3.framework.symbols.windows.extensions), 465
 TOKEN.VolTemplateProxy (class in volatility3.framework.symbols.windows.extensions), 465
 translate() (Bytes method), 264
 translate() (Elf64Layer method), 193
 translate() (HexBytes method), 312
 translate() (Intel method), 197
 translate() (Intel32e method), 200
 translate() (IntelPAE method), 202
 translate() (LimeLayer method), 216
 translate() (LinearlyMappedLayer method), 219
 translate() (MultiTypeData method), 316
 translate() (PdbMSFStream method), 222
 translate() (PdbMultiStreamFormat method), 225
 translate() (RegistryHive method), 236
 translate() (SegmentedLayer method), 243
 translate() (String method), 292
 translate() (VmwareLayer method), 245
 translate() (WindowsCrashDump32Layer method), 187
 translate() (WindowsCrashDump64Layer method), 190
 translate() (WindowsIntel method), 205
 translate() (WindowsIntel32e method), 208
 translate() (WindowsIntelPAE method), 210
 translate() (WindowsMixin method), 213
 translate() (XenCoreDumpLayer method), 248
 TranslationLayerInterface (class in volatility3.framework.interfaces.layers), 158
 TranslationLayerRequirement (class in volatility3.framework.configuration.requirements), 114
 traverse() (MMVAD method), 457
 traverse() (MMVAD_SHORT method), 460
 traverse() (SERVICE_RECORD method), 516
 TreeGrid (class in volatility3.framework.interfaces.renderers), 169
 TreeGrid (class in volatility3.framework.renderers), 302
 TreeNode (class in volatility3.framework.interfaces.renderers), 171
 TreeNode (class in volatility3.framework.renderers), 303
 truncate() (FileHandlerInterface method), 166
 truncate() (NullFileHandler method), 52

Trustedbsd (class in volatility3.plugins.mac.trustedbsd), 643
tty_check (class in volatility3.plugins.linux.tty_check), 605
type (Column attribute), 169
type (SymbolInterface property), 176
TYPE (SymbolType attribute), 319
type_handlers (PdbReader attribute), 519
type_name (SymbolInterface property), 176
type_prefix (elf_phdr property), 395
types (BaseSymbolTableInterface property), 173
types (BashIntermedSymbols property), 400
types (IntermediateSymbolTable property), 529
types (ISFormatTable property), 525
types (LinuxKernelIntermedSymbols property), 324
types (MacKernelIntermedSymbols property), 403
types (NativeTable property), 555
types (NativeTableInterface property), 175
types (SymbolTableInterface property), 180
types (Version1Format property), 532
types (Version2Format property), 535
types (Version3Format property), 537
types (Version4Format property), 540
types (Version5Format property), 543
types (Version6Format property), 546
types (Version7Format property), 549
types (Version8Format property), 552
types (WindowsKernelIntermedSymbols property), 432

U

uncompress() (in module volatility3.framework.layers.avml), 185
UNICODE_STRING (class in volatility3.framework.symbols.windows.extensions), 466
UNICODE_STRING.VolTemplateProxy (class in volatility3.framework.symbols.windows.extensions), 467
UnionType (class in volatility3.framework.objects), 294
UnionType.VolTemplateProxy (class in volatility3.framework.objects), 294
unix_sock (class in volatility3.framework.symbols.linux.extensions), 378
unix_sock.VolTemplateProxy (class in volatility3.framework.symbols.linux.extensions), 379
unixtime_to_datetime() (in module volatility3.framework.renderers.conversion), 305
UnparsableValue (class in volatility3.framework.renderers), 304
UnreadableValue (class in volatility3.framework.renderers), 304
unsatisfied() (ADS class method), 703

unsatisfied() (AutomagicInterface class method), 136
unsatisfied() (AVMLLayer class method), 184
unsatisfied() (Banners class method), 755
unsatisfied() (Bash class method), 560, 609
unsatisfied() (BashIntermedSymbols class method), 400
unsatisfied() (BigPools class method), 657
unsatisfied() (BooleanRequirement method), 96
unsatisfied() (BufferDataLayer class method), 227
unsatisfied() (BytesRequirement method), 98
unsatisfied() (Cachedump class method), 659
unsatisfied() (Callbacks class method), 662
unsatisfied() (Capabilities class method), 562
unsatisfied() (Check_afinfo class method), 564
unsatisfied() (Check_creds class method), 566
unsatisfied() (Check_idt class method), 567
unsatisfied() (Check_modules class method), 569
unsatisfied() (Check_syscall class method), 571, 611
unsatisfied() (Check_sysctl class method), 613
unsatisfied() (Check_trap_table class method), 614
unsatisfied() (ChoiceRequirement method), 99
unsatisfied() (ClassRequirement method), 138
unsatisfied() (CmdLine class method), 664
unsatisfied() (ComplexListRequirement method), 101
unsatisfied() (ConfigurableInterface class method), 139
unsatisfied() (ConfigurableModule class method), 125
unsatisfied() (ConfigurableRequirementInterface method), 141
unsatisfied() (ConfigWriter class method), 756
unsatisfied() (ConstructableRequirementInterface method), 143
unsatisfied() (ConstructionMagic class method), 72
unsatisfied() (Crashinfo class method), 665
unsatisfied() (DataLayerInterface class method), 156
unsatisfied() (DeviceTree class method), 667
unsatisfied() (DllList class method), 669
unsatisfied() (DriverIrp class method), 670
unsatisfied() (DriverModule class method), 672
unsatisfied() (DriverScan class method), 674
unsatisfied() (DumpFiles class method), 676
unsatisfied() (Elf64Layer class method), 193
unsatisfied() (Elfs class method), 573
unsatisfied() (Envars class method), 574, 678
unsatisfied() (FileLayer class method), 229
unsatisfied() (FileScan class method), 680
unsatisfied() (FrameworkInfo class method), 758
unsatisfied() (GetServiceSIDs class method), 681
unsatisfied() (GetSIDs class method), 683
unsatisfied() (Handles class method), 686
unsatisfied() (Hashdump class method), 688
unsatisfied() (HiveList class method), 649
unsatisfied() (HiveScan class method), 651

- `unsatisfied()` (*Ifconfig* class method), 616
- `unsatisfied()` (*Info* class method), 691
- `unsatisfied()` (*Intel* class method), 197
- `unsatisfied()` (*Intel32e* class method), 200
- `unsatisfied()` (*IntelPAE* class method), 202
- `unsatisfied()` (*IntermediateSymbolTable* class method), 529
- `unsatisfied()` (*IntRequirement* method), 102
- `unsatisfied()` (*IOMem* class method), 576
- `unsatisfied()` (*IsfInfo* class method), 759
- `unsatisfied()` (*ISFormatTable* class method), 525
- `unsatisfied()` (*JobLinks* class method), 692
- `unsatisfied()` (*Kauth_listeners* class method), 617
- `unsatisfied()` (*Kauth_scopes* class method), 619
- `unsatisfied()` (*KernelModule* class method), 78
- `unsatisfied()` (*KernelPDBScanner* class method), 81
- `unsatisfied()` (*Kevents* class method), 621
- `unsatisfied()` (*Keyboard_notifiers* class method), 578
- `unsatisfied()` (*Kmsg* class method), 581
- `unsatisfied()` (*LayerListRequirement* method), 104
- `unsatisfied()` (*LayerStacker* class method), 84
- `unsatisfied()` (*LayerWriter* class method), 761
- `unsatisfied()` (*LdrModules* class method), 694
- `unsatisfied()` (*LimeLayer* class method), 216
- `unsatisfied()` (*LinearlyMappedLayer* class method), 219
- `unsatisfied()` (*LinuxKernelIntermedSymbols* class method), 324
- `unsatisfied()` (*LinuxSymbolFinder* class method), 74
- `unsatisfied()` (*List_Files* class method), 623
- `unsatisfied()` (*ListRequirement* method), 105
- `unsatisfied()` (*Lsadbump* class method), 696
- `unsatisfied()` (*Lsmdb* class method), 586, 625
- `unsatisfied()` (*Lsof* class method), 588, 626
- `unsatisfied()` (*MacKernelIntermedSymbols* class method), 404
- `unsatisfied()` (*MacSymbolFinder* class method), 76
- `unsatisfied()` (*Malfind* class method), 589, 628, 698
- `unsatisfied()` (*Maps* class method), 594, 633
- `unsatisfied()` (*MBRScan* class method), 700
- `unsatisfied()` (*Memmap* class method), 701
- `unsatisfied()` (*MFTScan* class method), 704
- `unsatisfied()` (*ModScan* class method), 707
- `unsatisfied()` (*Module* class method), 130
- `unsatisfied()` (*ModuleInterface* class method), 153
- `unsatisfied()` (*ModuleRequirement* method), 107
- `unsatisfied()` (*Modules* class method), 709
- `unsatisfied()` (*Mount* class method), 630
- `unsatisfied()` (*MountInfo* class method), 591
- `unsatisfied()` (*MultiRequirement* method), 109
- `unsatisfied()` (*MutantScan* class method), 711
- `unsatisfied()` (*NetScan* class method), 714
- `unsatisfied()` (*NetStat* class method), 719
- `unsatisfied()` (*Netstat* class method), 631
- `unsatisfied()` (*NonLinearlySegmentedLayer* class method), 241
- `unsatisfied()` (*PdbMSFStream* class method), 222
- `unsatisfied()` (*PdbMultiStreamFormat* class method), 225
- `unsatisfied()` (*PluginInterface* class method), 168
- `unsatisfied()` (*PluginRequirement* method), 110
- `unsatisfied()` (*PoolScanner* class method), 722
- `unsatisfied()` (*PrintKey* class method), 653
- `unsatisfied()` (*Privs* class method), 725
- `unsatisfied()` (*PsAux* class method), 596
- `unsatisfied()` (*Psaux* class method), 635
- `unsatisfied()` (*PsList* class method), 598, 638, 728
- `unsatisfied()` (*PsScan* class method), 600, 730
- `unsatisfied()` (*PsTree* class method), 602, 640, 732
- `unsatisfied()` (*QemuSuspendLayer* class method), 233
- `unsatisfied()` (*RegistryHive* class method), 236
- `unsatisfied()` (*RequirementInterface* method), 146
- `unsatisfied()` (*SegmentedLayer* class method), 243
- `unsatisfied()` (*Sessions* class method), 734
- `unsatisfied()` (*SimpleTypeRequirement* method), 147
- `unsatisfied()` (*SizedModule* class method), 133
- `unsatisfied()` (*Skeleton_Key_Check* class method), 735
- `unsatisfied()` (*Socket_filters* class method), 641
- `unsatisfied()` (*Sockstat* class method), 604
- `unsatisfied()` (*SSDT* class method), 737
- `unsatisfied()` (*StringRequirement* method), 112
- `unsatisfied()` (*Strings* class method), 739
- `unsatisfied()` (*SvcScan* class method), 741
- `unsatisfied()` (*SymbolCacheMagic* class method), 90
- `unsatisfied()` (*SymbolFinder* class method), 92
- `unsatisfied()` (*SymbolTableInterface* class method), 180
- `unsatisfied()` (*SymbolTableRequirement* method), 114
- `unsatisfied()` (*SymlinkScan* class method), 743
- `unsatisfied()` (*Timeliner* class method), 765
- `unsatisfied()` (*Timers* class method), 643
- `unsatisfied()` (*TranslationLayerInterface* class method), 160
- `unsatisfied()` (*TranslationLayerRequirement* method), 115
- `unsatisfied()` (*Trustedbsd* class method), 645
- `unsatisfied()` (*tty_check* class method), 606
- `unsatisfied()` (*URIRequirement* method), 117
- `unsatisfied()` (*UserAssist* class method), 655
- `unsatisfied()` (*VadInfo* class method), 745
- `unsatisfied()` (*VadWalk* class method), 747
- `unsatisfied()` (*VadYaraScan* class method), 749
- `unsatisfied()` (*VerInfo* class method), 751
- `unsatisfied()` (*Version1Format* class method), 532
- `unsatisfied()` (*Version2Format* class method), 535
- `unsatisfied()` (*Version3Format* class method), 537
- `unsatisfied()` (*Version4Format* class method), 540

- `unsatisfied()` (*Version5Format* class method), 543
 - `unsatisfied()` (*Version6Format* class method), 546
 - `unsatisfied()` (*Version7Format* class method), 549
 - `unsatisfied()` (*Version8Format* class method), 552
 - `unsatisfied()` (*VersionRequirement* method), 118
 - `unsatisfied()` (*VFSevents* class method), 646
 - `unsatisfied()` (*VirtMap* class method), 753
 - `unsatisfied()` (*VmaYaraScan* class method), 608
 - `unsatisfied()` (*VmwareLayer* class method), 245
 - `unsatisfied()` (*Volshell* class method), 55, 58, 60, 63
 - `unsatisfied()` (*WindowsCrashDump32Layer* class method), 187
 - `unsatisfied()` (*WindowsCrashDump64Layer* class method), 190
 - `unsatisfied()` (*WindowsIntel* class method), 205
 - `unsatisfied()` (*WindowsIntel32e* class method), 208
 - `unsatisfied()` (*WindowsIntelPAE* class method), 211
 - `unsatisfied()` (*WindowsKernelIntermedSymbols* class method), 432
 - `unsatisfied()` (*WindowsMixin* class method), 213
 - `unsatisfied()` (*WinSwapLayers* class method), 94
 - `unsatisfied()` (*XenCoreDumpLayer* class method), 248
 - `unsatisfied()` (*YaraScan* class method), 767
 - `unsatisfied_children()` (*BooleanRequirement* method), 96
 - `unsatisfied_children()` (*BytesRequirement* method), 98
 - `unsatisfied_children()` (*ChoiceRequirement* method), 99
 - `unsatisfied_children()` (*ClassRequirement* method), 138
 - `unsatisfied_children()` (*ComplexListRequirement* method), 101
 - `unsatisfied_children()` (*ConfigurableRequirementInterface* method), 141
 - `unsatisfied_children()` (*ConstructableRequirementInterface* method), 143
 - `unsatisfied_children()` (*IntRequirement* method), 102
 - `unsatisfied_children()` (*LayerListRequirement* method), 104
 - `unsatisfied_children()` (*ListRequirement* method), 106
 - `unsatisfied_children()` (*ModuleRequirement* method), 107
 - `unsatisfied_children()` (*MultiRequirement* method), 109
 - `unsatisfied_children()` (*PluginRequirement* method), 111
 - `unsatisfied_children()` (*RequirementInterface* method), 146
 - `unsatisfied_children()` (*SimpleTypeRequirement* method), 147
 - `unsatisfied_children()` (*StringRequirement* method), 112
 - `unsatisfied_children()` (*SymbolTableRequirement* method), 114
 - `unsatisfied_children()` (*TranslationLayerRequirement* method), 116
 - `unsatisfied_children()` (*URIRequirement* method), 117
 - `unsatisfied_children()` (*VersionRequirement* method), 119
 - `UnsatisfiedException`, 558
 - `update()` (*CacheManagerInterface* method), 86
 - `update()` (*SqliteCache* method), 88
 - `update_vol()` (*ObjectTemplate* method), 299
 - `update_vol()` (*ReferenceTemplate* method), 300
 - `update_vol()` (*SymbolSpace.UnresolvedTemplate* method), 317
 - `update_vol()` (*Template* method), 164
 - `upper()` (*Bytes* method), 264
 - `upper()` (*HexBytes* method), 312
 - `upper()` (*MultiTypeData* method), 316
 - `upper()` (*String* method), 292
 - `URIRequirement` (class in *volatility3.framework.configuration.requirements*), 116
 - `UserAssist` (class in *volatility3.plugins.windows.registry.userassist*), 654
 - `uses_cache()` (*ResourceAccessor* method), 238
- ## V
- `VACB` (class in *volatility3.framework.symbols.windows.extensions*), 468
 - `VACB.VolTemplateProxy` (class in *volatility3.framework.symbols.windows.extensions*), 469
 - `VACB_ARRAY` (*SHARED_CACHE_MAP* attribute), 462
 - `VACB_BLOCK` (*SHARED_CACHE_MAP* attribute), 462
 - `VACB_LEVEL_SHIFT` (*SHARED_CACHE_MAP* attribute), 462
 - `VACB_OFFSET_SHIFT` (*SHARED_CACHE_MAP* attribute), 462
 - `VACB_SIZE_OF_FIRST_LEVEL` (*SHARED_CACHE_MAP* attribute), 462
 - `vad_dump()` (*VadInfo* class method), 746
 - `VadInfo` (class in *volatility3.plugins.windows.vadinfo*), 744
 - `VadWalk` (class in *volatility3.plugins.windows.vadwalk*), 746
 - `VadYaraScan` (class in *volatility3.plugins.windows.vadyarascan*), 748
 - `valid()` (in module *volatility3.schemas*), 767
 - `validate()` (in module *volatility3.schemas*), 767

VALIDDUMP (*WindowsCrashDump32Layer attribute*), 185
 VALIDDUMP (*WindowsCrashDump64Layer attribute*), 188
 values (*TreeNode property*), 172, 304
 values() (*HierarchicalDict method*), 145
 values() (*LayerContainer method*), 157
 values() (*ModuleCollection method*), 131
 values() (*ModuleContainer method*), 150
 values() (*ObjectInformation method*), 161
 values() (*ReadOnlyMapping method*), 163
 values() (*SymbolSpace method*), 319
 values() (*SymbolSpaceInterface method*), 177
 values() (*TreeGrid method*), 170, 303
 VerInfo (*class in volatility3.plugins.windows.verinfo*), 750
 version (*ADS attribute*), 703
 version (*Banners attribute*), 755
 version (*Bash attribute*), 560, 610
 version (*BigPools attribute*), 657
 version (*BytesScanner attribute*), 180
 version (*Cachedump attribute*), 659
 version (*CacheManagerInterface attribute*), 86
 version (*Callbacks attribute*), 662
 version (*Capabilities attribute*), 562
 version (*Check_afinfo attribute*), 564
 version (*Check_creds attribute*), 566
 version (*Check_idt attribute*), 568
 version (*Check_modules attribute*), 569
 version (*Check_syscall attribute*), 571, 611
 version (*Check_sysctl attribute*), 613
 version (*Check_trap_table attribute*), 615
 version (*CmdLine attribute*), 664
 version (*ConfigWriter attribute*), 756
 version (*Crashinfo attribute*), 665
 version (*DeviceTree attribute*), 667
 version (*DllList attribute*), 669
 version (*DriverIrp attribute*), 671
 version (*DriverModule attribute*), 672
 version (*DriverScan attribute*), 674
 version (*DumpFiles attribute*), 677
 version (*Elfs attribute*), 573
 version (*Envvars attribute*), 574, 678
 version (*FileScan attribute*), 680
 version (*FrameworkInfo attribute*), 758
 version (*GetServiceSIDs attribute*), 682
 version (*GetSIDs attribute*), 683
 version (*Handles attribute*), 686
 version (*Hashdump attribute*), 689
 version (*HiveList attribute*), 649
 version (*HiveScan attribute*), 651
 version (*Ifconfig attribute*), 616
 version (*Info attribute*), 691
 version (*IOMem attribute*), 576
 version (*IsInfo attribute*), 760
 version (*ISFormatTable attribute*), 525
 version (*JobLinks attribute*), 693
 version (*Kauth_listeners attribute*), 618
 version (*Kauth_scopes attribute*), 619
 version (*Kevents attribute*), 622
 version (*Keyboard_notifiers attribute*), 578
 version (*Kmsg attribute*), 581
 version (*LayerWriter attribute*), 761
 version (*LdrModules attribute*), 694
 VERSION (*LimeLayer attribute*), 214
 version (*LinuxUtilities attribute*), 326
 version (*List_Files attribute*), 623
 version (*Lsadump attribute*), 696
 version (*Lsmmod attribute*), 586, 625
 version (*Lsof attribute*), 588, 627
 version (*MacUtilities attribute*), 404
 version (*Malfind attribute*), 590, 628, 698
 version (*Maps attribute*), 594, 633
 version (*MBRScan attribute*), 700
 version (*Memmap attribute*), 701
 version (*MFTScan attribute*), 705
 version (*ModScan attribute*), 707
 version (*Modules attribute*), 710
 version (*Mount attribute*), 630
 version (*MountInfo attribute*), 591
 version (*MultiStringScanner attribute*), 181
 version (*MutantScan attribute*), 711
 version (*NetScan attribute*), 714
 version (*NetStat attribute*), 719
 version (*Netstat attribute*), 632
 version (*PageMapScanner attribute*), 93
 version (*PdbSignatureScanner attribute*), 522
 version (*PDBUtility attribute*), 521
 version (*PluginInterface attribute*), 168
 version (*PoolHeaderScanner attribute*), 719
 version (*PoolScanner attribute*), 722
 version (*PrintKey attribute*), 653
 version (*Privs attribute*), 725
 version (*PsAux attribute*), 596
 version (*Psaux attribute*), 635
 version (*PsList attribute*), 598, 638, 728
 version (*PsScan attribute*), 600, 731
 version (*PsTree attribute*), 602, 640, 733
 version (*RegExScanner attribute*), 181
 version (*ScannerInterface attribute*), 158
 version (*Sessions attribute*), 734
 version (*Skeleton_Key_Check attribute*), 736
 version (*Socket_filters attribute*), 642
 version (*SockHandlers attribute*), 603
 version (*Sockstat attribute*), 604
 version (*SqliteCache attribute*), 88
 version (*SSDT attribute*), 738
 version (*Strings attribute*), 740
 version (*SvcScan attribute*), 742
 version (*SymlinkScan attribute*), 743

- `version` (*Timeliner* attribute), 765
- `version` (*Timers* attribute), 643
- `version` (*Trustedbsd* attribute), 645
- `version` (*tty_check* attribute), 606
- `version` (*UserAssist* attribute), 655
- `version` (*VadInfo* attribute), 746
- `version` (*VadWalk* attribute), 748
- `version` (*VadYaraScan* attribute), 749
- `version` (*VerInfo* attribute), 751
- `version` (*Version1Format* attribute), 532
- `version` (*Version2Format* attribute), 535
- `version` (*Version3Format* attribute), 538
- `version` (*Version4Format* attribute), 541
- `version` (*Version5Format* attribute), 544
- `version` (*Version6Format* attribute), 547
- `version` (*Version7Format* attribute), 550
- `version` (*Version8Format* attribute), 553
- `version` (*VersionableInterface* attribute), 148
- `version` (*VFSevents* attribute), 646
- `version` (*VirtMap* attribute), 753
- `version` (*VmaYaraScan* attribute), 608
- `version` (*Volshell* attribute), 55, 58, 61, 64
- `version` (*YaraScan* attribute), 767
- `version` (*YaraScanner* attribute), 767
- `Version1Format` (class in `volatility3.framework.symbols.intermed`), 529
- `Version2Format` (class in `volatility3.framework.symbols.intermed`), 532
- `Version3Format` (class in `volatility3.framework.symbols.intermed`), 535
- `Version4Format` (class in `volatility3.framework.symbols.intermed`), 538
- `Version5Format` (class in `volatility3.framework.symbols.intermed`), 541
- `Version6Format` (class in `volatility3.framework.symbols.intermed`), 544
- `Version7Format` (class in `volatility3.framework.symbols.intermed`), 547
- `Version8Format` (class in `volatility3.framework.symbols.intermed`), 550
- `VersionableInterface` (class in `volatility3.framework.interfaces.configuration`), 148
- `VersionRequirement` (class in `volatility3.framework.configuration.requirements`), 117
- `VFSevents` (class in `volatility3.plugins.mac.vfsevents`), 645
- `vfsmount` (class in `volatility3.framework.symbols.linux.extensions`), 380
- `vfsmount.VolTemplateProxy` (class in `volatility3.framework.symbols.linux.extensions`), 380
- `VirtMap` (class in `volatility3.plugins.windows.virtmap`), 752
- `virtual_process_from_physical()` (*PsScan* class method), 731
- `virtual_to_physical_address()` (*LinuxIntelStacker* class method), 72
- `virtual_to_physical_address()` (*MacIntelStacker* class method), 74
- `visit()` (*TreeGrid* method), 171, 303
- `visit_nodes()` (*RegistryHive* method), 236
- `vm_area_struct` (class in `volatility3.framework.symbols.linux.extensions`), 384
- `vm_area_struct.VolTemplateProxy` (class in `volatility3.framework.symbols.linux.extensions`), 384
- `vm_map_entry` (class in `volatility3.framework.symbols.mac.extensions`), 424
- `vm_map_entry.VolTemplateProxy` (class in `volatility3.framework.symbols.mac.extensions`), 424
- `vm_map_object` (class in `volatility3.framework.symbols.mac.extensions`), 426
- `vm_map_object.VolTemplateProxy` (class in `volatility3.framework.symbols.mac.extensions`), 426
- `vma_dump()` (*Maps* class method), 594
- `VmaYaraScan` (class in `volatility3.plugins.linux.vmayarascan`), 606
- `VmwareFormatException`, 243
- `VmwareLayer` (class in `volatility3.framework.layers.vmware`), 243
- `VmwareStacker` (class in `volatility3.framework.layers.vmware`), 246
- `vnode` (class in `volatility3.framework.symbols.mac.extensions`), 427
- `vnode.VolTemplateProxy` (class in `volatility3.framework.symbols.mac.extensions`), 427
- `vnode_filters` (*Kevents* attribute), 622
- `Void` (class in `volatility3.framework.objects`), 296
- `Void.VolTemplateProxy` (class in `volatility3.framework.objects`), 296
- `vol` (*AggregateType* property), 251
- `vol` (*Array* property), 253
- `vol` (*BitField* property), 256
- `vol` (*Boolean* property), 259
- `vol` (*bpf_prog* property), 328
- `vol` (*bt_sock* property), 329
- `vol` (*Bytes* property), 264
- `vol` (*Char* property), 268
- `vol` (*ClassType* property), 269
- `vol` (*CM_KEY_BODY* property), 504
- `vol` (*CM_KEY_NODE* property), 506

vol (*CM_KEY_VALUE* property), 508
 vol (*CMHIVE* property), 502
 vol (*CONTROL_AREA* property), 434
 vol (*cred* property), 331
 vol (*dentry* property), 333
 vol (*DEVICE_OBJECT* property), 437
 vol (*DRIVER_OBJECT* property), 439
 vol (*elf* property), 393
 vol (*elf_phdr* property), 395
 vol (*elf_sym* property), 397
 vol (*Enumeration* property), 273
 vol (*EPROCESS* property), 441
 vol (*ETHREAD* property), 443
 vol (*EX_FAST_REF* property), 445
 vol (*ExecutiveObject* property), 491
 vol (*FILE_OBJECT* property), 447
 vol (*fileglob* property), 406
 vol (*files_struct* property), 335
 vol (*Float* property), 276
 vol (*fs_struct* property), 337
 vol (*Function* property), 277
 vol (*GenericIntelProcess* property), 321
 vol (*hist_entry* property), 391
 vol (*HMAP_ENTRY* property), 510
 vol (*ifnet* property), 408
 vol (*IMAGE_DOS_HEADER* property), 487
 vol (*IMAGE_NT_HEADERS* property), 489
 vol (*inet_sock* property), 339
 vol (*inpcb* property), 410
 vol (*Integer* property), 281
 vol (*kauth_scope* property), 412
 vol (*KDDEBUGGER_DATA64* property), 474
 vol (*kernel_cap_struct* property), 342
 vol (*kernel_cap_t* property), 344
 vol (*KMUTANT* property), 449
 vol (*kobject* property), 346
 vol (*KSYSTEM_TIME* property), 451
 vol (*KTHREAD* property), 453
 vol (*LIST_ENTRY* property), 455
 vol (*list_head* property), 348
 vol (*maple_tree* property), 350
 vol (*MFTAttribute* property), 481
 vol (*MFTEntry* property), 482
 vol (*MFTFileName* property), 484
 vol (*mm_struct* property), 352
 vol (*MMVAD* property), 457
 vol (*MMVAD_SHORT* property), 460
 vol (*mnt_namespace* property), 354
 vol (*module* property), 357
 vol (*mount* property), 360
 vol (*net* property), 362
 vol (*netlink_sock* property), 364
 vol (*OBJECT_HEADER* property), 493
 vol (*OBJECT_SYMBOLIC_LINK* property), 462
 vol (*ObjectInterface* property), 163
 vol (*ObjectTemplate* property), 299
 vol (*packet_sock* property), 366
 vol (*PARTITION_ENTRY* property), 477
 vol (*PARTITION_TABLE* property), 479
 vol (*Pointer* property), 284
 vol (*POOL_HEADER* property), 495
 vol (*POOL_HEADER_VISTA* property), 498
 vol (*POOL_TRACKER_BIG_PAGES* property), 500
 vol (*PrimitiveObject* property), 286
 vol (*proc* property), 414
 vol (*qstr* property), 368
 vol (*queue_entry* property), 416
 vol (*ReferenceTemplate* property), 300
 vol (*SERVICE_HEADER* property), 514
 vol (*SERVICE_RECORD* property), 516
 vol (*SHARED_CACHE_MAP* property), 464
 vol (*sock* property), 370
 vol (*sockaddr* property), 418
 vol (*sockaddr_dl* property), 419
 vol (*socket* property), 371, 421
 vol (*String* property), 292
 vol (*struct_file* property), 373
 vol (*StructType* property), 294
 vol (*SUMMARY_DUMP* property), 472
 vol (*super_block* property), 376
 vol (*SymbolSpace.UnresolvedTemplate* property), 317
 vol (*sysctl_oid* property), 423
 vol (*task_struct* property), 378
 vol (*Template* property), 164
 vol (*TOKEN* property), 466
 vol (*UNICODE_STRING* property), 468
 vol (*UnionType* property), 296
 vol (*unix_sock* property), 380
 vol (*VACB* property), 470
 vol (*vfsmount* property), 383
 vol (*vm_area_struct* property), 386
 vol (*vm_map_entry* property), 425
 vol (*vm_map_object* property), 427
 vol (*vnode* property), 429
 vol (*Void* property), 297
 vol (*vsock_sock* property), 387
 vol (*xdp_sock* property), 389
 volatility3
 module, 47
 volatility3.cli
 module, 48
 volatility3.cli.text_renderer
 module, 64
 volatility3.cli.volargparse
 module, 67
 volatility3.cli.volshell
 module, 49
 volatility3.cli.volshell.generic

- module, 50
- volatility3.cli.volshell.linux
 - module, 55
- volatility3.cli.volshell.mac
 - module, 58
- volatility3.cli.volshell.windows
 - module, 61
- volatility3.framework
 - module, 68
- volatility3.framework.automagic
 - module, 69
- volatility3.framework.automagic.construct_layers
 - module, 70
- volatility3.framework.automagic.linux
 - module, 72
- volatility3.framework.automagic.mac
 - module, 74
- volatility3.framework.automagic.module
 - module, 77
- volatility3.framework.automagic.pdbscan
 - module, 78
- volatility3.framework.automagic.stacker
 - module, 82
- volatility3.framework.automagic.symbol_cache
 - module, 85
- volatility3.framework.automagic.symbol_finder
 - module, 90
- volatility3.framework.automagic.windows
 - module, 92
- volatility3.framework.configuration
 - module, 95
- volatility3.framework.configuration.requirements
 - module, 95
- volatility3.framework.constants
 - module, 119
- volatility3.framework.constants.linux
 - module, 122
- volatility3.framework.constants.windows
 - module, 122
- volatility3.framework.contexts
 - module, 123
- volatility3.framework.exceptions
 - module, 555
- volatility3.framework.interfaces
 - module, 134
- volatility3.framework.interfaces.automagic
 - module, 134
- volatility3.framework.interfaces.configuration
 - module, 137
- volatility3.framework.interfaces.context
 - module, 148
- volatility3.framework.interfaces.layers
 - module, 154
- volatility3.framework.interfaces.objects
 - module, 161
- volatility3.framework.interfaces.plugins
 - module, 165
- volatility3.framework.interfaces.renderers
 - module, 168
- volatility3.framework.interfaces.symbols
 - module, 172
- volatility3.framework.layers
 - module, 180
- volatility3.framework.layers.avml
 - module, 182
- volatility3.framework.layers.cloudstorage
 - module, 185
- volatility3.framework.layers.codecs
 - module, 180
- volatility3.framework.layers.crash
 - module, 185
- volatility3.framework.layers.elf
 - module, 191
- volatility3.framework.layers.intel
 - module, 195
- volatility3.framework.layers.leechcore
 - module, 214
- volatility3.framework.layers.lime
 - module, 214
- volatility3.framework.layers.linear
 - module, 217
- volatility3.framework.layers.msf
 - module, 220
- volatility3.framework.layers.physical
 - module, 225
- volatility3.framework.layers.qemu
 - module, 230
- volatility3.framework.layers.registry
 - module, 233
- volatility3.framework.layers.resources
 - module, 237
- volatility3.framework.layers.scanners
 - module, 180
- volatility3.framework.layers.scanners.multiregexp
 - module, 181
- volatility3.framework.layers.segmented
 - module, 239
- volatility3.framework.layers.vmware
 - module, 243
- volatility3.framework.layers.xen
 - module, 246
- volatility3.framework.objects
 - module, 249
- volatility3.framework.objects.templates
 - module, 298
- volatility3.framework.objects.utility
 - module, 300
- volatility3.framework.plugins

module, 301
 volatility3.framework.renderers
 module, 301
 volatility3.framework.renderers.conversion
 module, 304
 volatility3.framework.renderers.format_hints
 module, 305
 volatility3.framework.symbols
 module, 316
 volatility3.framework.symbols.generic
 module, 319
 volatility3.framework.symbols.intermed
 module, 523
 volatility3.framework.symbols.linux
 module, 321
 volatility3.framework.symbols.linux.bash
 module, 397
 volatility3.framework.symbols.linux.extensions
 module, 326
 volatility3.framework.symbols.linux.extensions.volatile
 module, 390
 volatility3.framework.symbols.linux.extensions.volatile
 module, 392
 volatility3.framework.symbols.mac
 module, 401
 volatility3.framework.symbols.mac.extensions
 module, 405
 volatility3.framework.symbols.metadata
 module, 553
 volatility3.framework.symbols.native
 module, 553
 volatility3.framework.symbols.windows
 module, 429
 volatility3.framework.symbols.windows.extensions
 module, 432
 volatility3.framework.symbols.windows.extensions.volatile
 module, 470
 volatility3.framework.symbols.windows.extensions.volatile
 module, 472
 volatility3.framework.symbols.windows.extensions.volatile
 module, 474
 volatility3.framework.symbols.windows.extensions.volatile
 module, 479
 volatility3.framework.symbols.windows.extensions.volatile
 module, 485
 volatility3.framework.symbols.windows.extensions.volatile
 module, 485
 volatility3.framework.symbols.windows.extensions.volatile
 module, 489
 volatility3.framework.symbols.windows.extensions.volatile
 module, 500
 volatility3.framework.symbols.windows.extensions.volatile
 module, 512
 volatility3.framework.symbols.windows.pdbconv
 module, 517
 volatility3.framework.symbols.windows.pdbutil
 module, 519
 volatility3.framework.symbols.windows.versions
 module, 522
 volatility3.framework.symbols.wrappers
 module, 555
 volatility3.plugins
 module, 558
 volatility3.plugins.banners
 module, 753
 volatility3.plugins.configwriter
 module, 755
 volatility3.plugins.frameworkinfo
 module, 757
 volatility3.plugins.isfinfo
 module, 758
 volatility3.plugins.layerwriter
 module, 760
 volatility3.plugins.linux
 module, 558
 volatility3.plugins.linux.bash
 module, 559
 volatility3.plugins.linux.capabilities
 module, 560
 volatility3.plugins.linux.check_afinfo
 module, 563
 volatility3.plugins.linux.check_creds
 module, 565
 volatility3.plugins.linux.check_idt
 module, 566
 volatility3.plugins.linux.check_modules
 module, 568
 volatility3.plugins.linux.check_syscall
 module, 569
 volatility3.plugins.linux.elfs
 module, 571
 volatility3.plugins.linux.envvars
 module, 573
 volatility3.plugins.linux.iomem
 module, 575
 volatility3.plugins.linux.keyboard_notifiers
 module, 577
 volatility3.plugins.linux.kmsg
 module, 578
 volatility3.plugins.linux.lsmod
 module, 585
 volatility3.plugins.linux.lsof
 module, 587
 volatility3.plugins.linux.malfind
 module, 588
 volatility3.plugins.linux.mountinfo
 module, 590
 volatility3.plugins.linux.proc

- module, 592
- volatility3.plugins.linux.psaux
 - module, 595
- volatility3.plugins.linux.pslist
 - module, 596
- volatility3.plugins.linux.psscan
 - module, 599
- volatility3.plugins.linux.pstree
 - module, 601
- volatility3.plugins.linux.sockstat
 - module, 602
- volatility3.plugins.linux.tty_check
 - module, 605
- volatility3.plugins.linux.vmayarascan
 - module, 606
- volatility3.plugins.mac
 - module, 608
- volatility3.plugins.mac.bash
 - module, 608
- volatility3.plugins.mac.check_syscall
 - module, 610
- volatility3.plugins.mac.check_sysctl
 - module, 612
- volatility3.plugins.mac.check_trap_table
 - module, 613
- volatility3.plugins.mac.ifconfig
 - module, 615
- volatility3.plugins.mac.kauth_listeners
 - module, 616
- volatility3.plugins.mac.kauth_scopes
 - module, 618
- volatility3.plugins.mac.kevents
 - module, 620
- volatility3.plugins.mac.list_files
 - module, 622
- volatility3.plugins.mac.lsmod
 - module, 623
- volatility3.plugins.mac.lsof
 - module, 625
- volatility3.plugins.mac.malfind
 - module, 627
- volatility3.plugins.mac.mount
 - module, 628
- volatility3.plugins.mac.netstat
 - module, 630
- volatility3.plugins.mac.proc_maps
 - module, 632
- volatility3.plugins.mac.psaux
 - module, 633
- volatility3.plugins.mac.pslist
 - module, 635
- volatility3.plugins.mac.pstree
 - module, 639
- volatility3.plugins.mac.socket_filters
 - module, 640
- volatility3.plugins.mac.timers
 - module, 642
- volatility3.plugins.mac.trustedbsd
 - module, 643
- volatility3.plugins.mac.vfsevents
 - module, 645
- volatility3.plugins.timeliner
 - module, 762
- volatility3.plugins.windows
 - module, 647
- volatility3.plugins.windows.bigpools
 - module, 655
- volatility3.plugins.windows.cachedump
 - module, 657
- volatility3.plugins.windows.callbacks
 - module, 659
- volatility3.plugins.windows.cmdline
 - module, 662
- volatility3.plugins.windows.crashinfo
 - module, 664
- volatility3.plugins.windows.devicetree
 - module, 666
- volatility3.plugins.windows.dlllist
 - module, 667
- volatility3.plugins.windows.driverirp
 - module, 669
- volatility3.plugins.windows.drivermodule
 - module, 671
- volatility3.plugins.windows.driverscan
 - module, 673
- volatility3.plugins.windows.dumpfiles
 - module, 675
- volatility3.plugins.windows.envars
 - module, 677
- volatility3.plugins.windows.filescan
 - module, 678
- volatility3.plugins.windows.getservicesids
 - module, 680
- volatility3.plugins.windows.getsids
 - module, 682
- volatility3.plugins.windows.handles
 - module, 684
- volatility3.plugins.windows.hashdump
 - module, 686
- volatility3.plugins.windows.info
 - module, 689
- volatility3.plugins.windows.joblinks
 - module, 691
- volatility3.plugins.windows.ldrmodules
 - module, 693
- volatility3.plugins.windows.lsadump
 - module, 694
- volatility3.plugins.windows.malfind

module, 696
 volatility3.plugins.windows.mbrscan
 module, 698
 volatility3.plugins.windows.memmap
 module, 700
 volatility3.plugins.windows.mftscan
 module, 702
 volatility3.plugins.windows.modscan
 module, 705
 volatility3.plugins.windows.modules
 module, 707
 volatility3.plugins.windows.mutantscan
 module, 710
 volatility3.plugins.windows.netscan
 module, 712
 volatility3.plugins.windows.netstat
 module, 714
 volatility3.plugins.windows.poolscanner
 module, 719
 volatility3.plugins.windows.privileges
 module, 724
 volatility3.plugins.windows.pslist
 module, 726
 volatility3.plugins.windows.psscan
 module, 729
 volatility3.plugins.windows.pstree
 module, 731
 volatility3.plugins.windows.registry
 module, 647
 volatility3.plugins.windows.registry.hivelist
 module, 647
 volatility3.plugins.windows.registry.hivescan
 module, 650
 volatility3.plugins.windows.registry.printkey
 module, 652
 volatility3.plugins.windows.registry.userassist
 module, 654
 volatility3.plugins.windows.sessions
 module, 733
 volatility3.plugins.windows.skeleton_key_check
 module, 734
 volatility3.plugins.windows.ssdt
 module, 736
 volatility3.plugins.windows.strings
 module, 738
 volatility3.plugins.windows.svcscan
 module, 740
 volatility3.plugins.windows.symlinkscan
 module, 742
 volatility3.plugins.windows.vadinfo
 module, 744
 volatility3.plugins.windows.vadwalk
 module, 746
 volatility3.plugins.windows.vadyarascan

module, 748
 volatility3.plugins.windows.verinfo
 module, 750
 volatility3.plugins.windows.virtmap
 module, 752
 volatility3.plugins.yarascan
 module, 765
 volatility3.schemas
 module, 767
 volatility3.symbols
 module, 768
 VolatilityException, 558
 VolatilityHandler (class in volatil-
 ity3.framework.layers.resources), 238
 VolShell (class in volatility3.cli.volshell), 49
 Volshell (class in volatility3.cli.volshell.generic), 52
 Volshell (class in volatility3.cli.volshell.linux), 55
 Volshell (class in volatility3.cli.volshell.mac), 58
 Volshell (class in volatility3.cli.volshell.windows), 61
 vsock_sock (class in volatil-
 ity3.framework.symbols.linux.extensions),
 386
 vsock_sock.VolTemplateProxy (class in volatil-
 ity3.framework.symbols.linux.extensions),
 386

W

walk_internal_list() (*LinuxUtilities* class method),
 326
 walk_list() (*queue_entry* method), 416
 walk_list_head() (*MacUtilities* class method), 404
 walk_slist() (*MacUtilities* class method), 404
 walk_tailq() (*MacUtilities* class method), 405
 WarningFindSpec (class in volatility3), 47
 WindowsCrashDump32Layer (class in volatil-
 ity3.framework.layers.crash), 185
 WindowsCrashDump64Layer (class in volatil-
 ity3.framework.layers.crash), 188
 WindowsCrashDumpFormatException, 190
 WindowsCrashDumpStacker (class in volatil-
 ity3.framework.layers.crash), 191
 WindowsIdentifier (class in volatil-
 ity3.framework.automagic.symbol_cache),
 90
 WindowsIntel (class in volatil-
 ity3.framework.layers.intel), 203
 WindowsIntel32e (class in volatil-
 ity3.framework.layers.intel), 205
 WindowsIntelPAE (class in volatil-
 ity3.framework.layers.intel), 208
 WindowsIntelStacker (class in volatil-
 ity3.framework.automagic.windows), 95
 WindowsKernelIntermedSymbols (class in volatil-
 ity3.framework.symbols.windows), 429

WindowsMetadata (class in volatility3.framework.symbols.metadata), 553

WindowsMixin (class in volatility3.framework.layers.intel), 211

WinSwapLayers (class in volatility3.framework.automagic.windows), 93

wintime_to_datetime() (in module volatility3.framework.renderers.conversion), 305

with_traceback() (ElfFormatException method), 194

with_traceback() (InvalidAddressException method), 556

with_traceback() (LayerException method), 556

with_traceback() (LimeFormatException method), 214

with_traceback() (MissingModuleException method), 556

with_traceback() (OfflineException method), 556

with_traceback() (PagedInvalidAddressException method), 556

with_traceback() (PDBFormatException method), 220

with_traceback() (PluginRequirementException method), 557

with_traceback() (PluginVersionException method), 557

with_traceback() (RegistryFormatException method), 233

with_traceback() (RegistryInvalidIndex method), 237

with_traceback() (SnappyException method), 185

with_traceback() (SwappedInvalidAddressException method), 557

with_traceback() (SymbolError method), 557

with_traceback() (SymbolSpaceError method), 558

with_traceback() (UnsatisfiedException method), 558

with_traceback() (VmwareFormatException method), 243

with_traceback() (VolatilityException method), 558

with_traceback() (WindowsCrashDumpFormatException method), 191

writable() (FileHandlerInterface method), 166

writable() (NullFileHandler method), 52

write() (AggregateType method), 251

write() (Array method), 253

write() (AVMLLayer method), 184

write() (BitField method), 256

write() (Boolean method), 259

write() (bpf_prog method), 328

write() (bt_sock method), 329

write() (BufferDataLayer method), 227

write() (Bytes method), 264

write() (Char method), 268

write() (ClassType method), 269

write() (CM_KEY_BODY method), 504

write() (CM_KEY_NODE method), 506

write() (CM_KEY_VALUE method), 508

write() (CMHIVE method), 502

write() (CONTROL_AREA method), 435

write() (cred method), 331

write() (DataLayerInterface method), 156

write() (dentry method), 333

write() (DEVICE_OBJECT method), 437

write() (DRIVER_OBJECT method), 439

write() (elf method), 393

write() (Elf64Layer method), 194

write() (elf_phdr method), 395

write() (elf_sym method), 397

write() (Enumeration method), 273

write() (EPROCESS method), 441

write() (ETHREAD method), 443

write() (EX_FAST_REF method), 445

write() (ExecutiveObject method), 491

write() (FILE_OBJECT method), 447

write() (fileglob method), 406

write() (FileHandlerInterface method), 166

write() (FileLayer method), 229

write() (files_struct method), 335

write() (Float method), 276

write() (fs_struct method), 337

write() (Function method), 277

write() (GenericIntelProcess method), 321

write() (hist_entry method), 391

write() (HMAP_ENTRY method), 510

write() (ifnet method), 408

write() (IMAGE_DOS_HEADER method), 487

write() (IMAGE_NT_HEADERS method), 489

write() (inet_sock method), 339

write() (inpcb method), 410

write() (Integer method), 281

write() (Intel method), 197

write() (Intel32e method), 200

write() (IntelPAE method), 203

write() (kauth_scope method), 412

write() (KDDEBUGGER_DATA64 method), 474

write() (kernel_cap_struct method), 342

write() (kernel_cap_t method), 345

write() (KMUTANT method), 449

write() (kobject method), 346

write() (KSYSTEM_TIME method), 451

write() (KTHREAD method), 453

write() (LayerContainer method), 158

write() (LimeLayer method), 216

write() (LinearlyMappedLayer method), 219

write() (LIST_ENTRY method), 455

write() (list_head method), 348

write() (maple_tree method), 350

write() (MFTAttribute method), 481

write() (MFTEntry method), 483

write() (MFTFileName method), 484

[write\(\) \(mm_struct method\), 352](#)
[write\(\) \(MMVAD method\), 457](#)
[write\(\) \(MMVAD_SHORT method\), 460](#)
[write\(\) \(mnt_namespace method\), 354](#)
[write\(\) \(module method\), 357](#)
[write\(\) \(mount method\), 360](#)
[write\(\) \(net method\), 362](#)
[write\(\) \(netlink_sock method\), 364](#)
[write\(\) \(NonLinearlySegmentedLayer method\), 241](#)
[write\(\) \(NullFileHandler method\), 52](#)
[write\(\) \(OBJECT_HEADER method\), 493](#)
[write\(\) \(OBJECT_SYMBOLIC_LINK method\), 462](#)
[write\(\) \(ObjectInterface method\), 163](#)
[write\(\) \(packet_sock method\), 366](#)
[write\(\) \(PARTITION_ENTRY method\), 477](#)
[write\(\) \(PARTITION_TABLE method\), 479](#)
[write\(\) \(PdbMSFStream method\), 222](#)
[write\(\) \(PdbMultiStreamFormat method\), 225](#)
[write\(\) \(Pointer method\), 284](#)
[write\(\) \(POOL_HEADER method\), 495](#)
[write\(\) \(POOL_HEADER_VISTA method\), 498](#)
[write\(\) \(POOL_TRACKER_BIG_PAGES method\), 500](#)
[write\(\) \(PrimitiveObject method\), 286](#)
[write\(\) \(proc method\), 414](#)
[write\(\) \(QemuSuspendLayer method\), 233](#)
[write\(\) \(qstr method\), 368](#)
[write\(\) \(queue_entry method\), 416](#)
[write\(\) \(RegistryHive method\), 236](#)
[write\(\) \(SegmentedLayer method\), 243](#)
[write\(\) \(SERVICE_HEADER method\), 514](#)
[write\(\) \(SERVICE_RECORD method\), 517](#)
[write\(\) \(SHARED_CACHE_MAP method\), 464](#)
[write\(\) \(sock method\), 370](#)
[write\(\) \(sockaddr method\), 418](#)
[write\(\) \(sockaddr_dl method\), 420](#)
[write\(\) \(socket method\), 371, 421](#)
[write\(\) \(String method\), 292](#)
[write\(\) \(struct_file method\), 373](#)
[write\(\) \(StructType method\), 294](#)
[write\(\) \(SUMMARY_DUMP method\), 472](#)
[write\(\) \(super_block method\), 376](#)
[write\(\) \(sysctl_oid method\), 424](#)
[write\(\) \(task_struct method\), 378](#)
[write\(\) \(TOKEN method\), 466](#)
[write\(\) \(TranslationLayerInterface method\), 160](#)
[write\(\) \(UNICODE_STRING method\), 468](#)
[write\(\) \(UnionType method\), 296](#)
[write\(\) \(unix_sock method\), 380](#)
[write\(\) \(VACB method\), 470](#)
[write\(\) \(vfsmount method\), 384](#)
[write\(\) \(vm_area_struct method\), 386](#)
[write\(\) \(vm_map_entry method\), 425](#)
[write\(\) \(vm_map_object method\), 427](#)
[write\(\) \(VmwareLayer method\), 246](#)

[write\(\) \(vnode method\), 429](#)
[write\(\) \(Void method\), 297](#)
[write\(\) \(vsock_sock method\), 388](#)
[write\(\) \(WindowsCrashDump32Layer method\), 187](#)
[write\(\) \(WindowsCrashDump64Layer method\), 190](#)
[write\(\) \(WindowsIntel method\), 205](#)
[write\(\) \(WindowsIntel32e method\), 208](#)
[write\(\) \(WindowsIntelPAE method\), 211](#)
[write\(\) \(WindowsMixin method\), 214](#)
[write\(\) \(xdp_sock method\), 389](#)
[write\(\) \(XenCoreDumpLayer method\), 248](#)
[write_layer\(\) \(LayerWriter class method\), 761](#)
[writelines\(\) \(FileHandlerInterface method\), 166](#)
[writelines\(\) \(NullFileHandler method\), 52](#)

X

[xdp_sock \(class in volatility3.framework.symbols.linux.extensions\), 388](#)
[xdp_sock.VolTemplateProxy \(class in volatility3.framework.symbols.linux.extensions\), 388](#)
[XenCoreDumpLayer \(class in volatility3.framework.layers.xen\), 246](#)
[XenCoreDumpStacker \(class in volatility3.framework.layers.xen\), 248](#)

Y

[YaraScan \(class in volatility3.plugins.yarascan\), 765](#)
[YaraScanner \(class in volatility3.plugins.yarascan\), 767](#)

Z

[zfill\(\) \(Bytes method\), 265](#)
[zfill\(\) \(HexBytes method\), 312](#)
[zfill\(\) \(MultiTypeData method\), 316](#)
[zfill\(\) \(String method\), 292](#)