

---

# **Volatility 3 Documentation**

***Release 2.0.0-beta.1***

**Volatility Foundation**

**Jan 25, 2021**



# CONTENTS

<b>1</b>	<b>Volatility 3 Basics</b>	<b>3</b>
1.1	Memory layers	3
1.2	Templates and Objects	4
1.3	Symbol Tables	4
1.4	Plugins	4
1.5	Output Renderers	5
1.6	Configuration Tree	5
1.7	Automagic	5
<b>2</b>	<b>How to Write a Simple Plugin</b>	<b>7</b>
2.1	Inherit from PluginInterface	7
2.2	Define the plugin requirements	7
2.3	Define the <i>run</i> method	9
2.4	Define the generator	10
<b>3</b>	<b>Changes between Volatility 2 and Volatility 3</b>	<b>13</b>
3.1	Library and Context	13
3.2	Symbols and Types	13
3.3	Object Model changes	13
3.4	Layer and Layer dependencies	14
3.5	Automagic	14
3.6	Searching and Scanning	14
3.7	Output Rendering	14
<b>4</b>	<b>Writing more advanced Plugins</b>	<b>15</b>
4.1	Writing Reusable Methods	15
4.2	Writing plugins that run other plugins	15
4.3	Writing plugins that output files	16
4.4	Writing Scanners	16
4.5	Writing/Using Intermediate Symbol Format Files	17
4.6	Writing new Translation Layers	18
4.7	Writing new Templates and Objects	20
<b>5</b>	<b>Using Volatility 3 as a Library</b>	<b>21</b>
5.1	Creating a context	21
5.2	Determine what plugins are available	21
5.3	Determine what configuration options a plugin requires	22
5.4	Set the configuration in the context	22
5.5	Using automagic to complete the configuration	23
5.6	Run the plugin	23

5.7	Render the TreeGrid . . . . .	24
<b>6</b>	<b>Creating New Symbol Tables</b>	<b>25</b>
6.1	How Volatility finds symbol tables . . . . .	25
6.2	Windows symbol tables . . . . .	25
6.3	Mac/Linux symbol tables . . . . .	26
<b>7</b>	<b>Python Packages</b>	<b>27</b>
7.1	volatility package . . . . .	27
<b>8</b>	<b>Indices and tables</b>	<b>545</b>
	<b>Python Module Index</b>	<b>547</b>
	<b>Index</b>	<b>551</b>

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*

Here are some guidelines for using Volatility 3 effectively:



## VOLATILITY 3 BASICS

Volatility splits memory analysis down to several components:

- 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. Memory is seen as sequential when accessed through sequential addresses, however, there is no obligation for the data to be stored sequentially, and modern processors tend to store the memory in a paged format. Moreover, there is no need for the data to be stored in an easily accessible format, it could be encoded or encrypted or more, it could be the combination of two other sources. These are typically handled by programs that process file formats, or the memory manager of the processor, but these are all translations (either in the geometric or linguistic sense) of the original data.

In Volatility 3 this is represented by a directed graph, whose end nodes are *DataLayers* and whose internal nodes are specifically called a *TranslationLayer*. 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 translated to determine where within the file the data is stored and that will be returned.

---

**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.

---



## 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.

### 2.1 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 volatility.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.

### 2.2 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.

```
@classmethod
def get_requirements(cls):
    return [requirements.TranslationLayerRequirement(name = 'primary',
                                                    description = 'Memory layer for_
↳ the kernel',
                                                    architectures = ["Intel32",
↳ "Intel64"]),
            requirements.SymbolTableRequirement(name = "nt_symbols",
                                                    description = "Windows kernel symbols
↳ "),
            requirements.PluginRequirement(name = 'pslist',
                                           plugin = pslist.PsList,
                                           version = (1, 0, 0)),
            requirements.ListRequirement(name = 'pid',
                                         element_type = int,
```

(continues on next page)

(continued from previous page)

```
description = "Process IDs to include (all_
↳other processes are excluded)",
optional = True)]
```

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.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.PluginRequirement(name = 'pslist',
plugin = pslist.PsList,
version = (1, 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.

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

The final 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.

## 2.3 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))

    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,
                                                    self.
↪config['primary'],
                                                    self.
↪config['nt_symbols'],
                                                    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 `volatility.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 `primary` and `nt_symbols` requirements. 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).

## 2.4 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:  
                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.





## CHANGES BETWEEN VOLATILITY 2 AND VOLATILITY 3

### 3.1 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.

### 3.2 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.

### 3.3 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.

## 3.4 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.

## 3.5 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.

## 3.6 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.

## 3.7 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.

## 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.

### 4.1 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.

### 4.2 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.

## 4.3 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).

## 4.4 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 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.

## 4.5 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 (volatility/symbols and volatility/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]
```

## 4.6 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 are surjective in that a single virtual address can map to multiple physical addresses, but a single virtual address can only ever map to a single 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 `bbbbbc` 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.

## 4.6.1 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.

---

## 4.7 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.



## 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*

### 5.1 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:

```
volatility.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
```

### 5.2 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, `volatility.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 `volatility.plugins` module. The standard paths should generally also be included, which can be found in `volatility.constants.PLUGINS_PATH`.

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

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()
```

## 5.3 Determine what configuration options a plugin requires

For each plugin class, we can call the classmethod `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).

## 5.4 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
```

## 5.5 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.

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.

## 5.6 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. A progress\_callback can also be provided to give users feedback whilst the plugin is running. 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.

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()
```

## 5.7 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.

### 6.1 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 `volatility/symbols`, underneath an operating system directory (currently one of `windows`, `mac` or `linux`). 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. The ZIP file must be named after the appropriate operating system (such as `linux.zip`, `mac.zip` or `windows.zip`). Inside the ZIP file, the directory structure should match the uncompressed operating system directory.

### 6.2 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 match the filename pattern of `<pdb-name>/<GUID>-<AGE>.json` (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 volatility/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.

## 6.3 Mac/Linux symbol tables

For Mac/Linux systems, both use the same mechanism for identification. JSON files live under the symbol directories, under either the `linux` or `mac` directories. 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 operating system 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. Once a kernel with debugging symbols/appropriate DWARF file has been located, `dwarf2json` will convert it into an appropriate JSON file.

## PYTHON PACKAGES

### 7.1 volatility package

Volatility 3 - An open-source memory forensics framework

**class WarningFindSpec**

Bases: `importlib.abc.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 change the deleter on a property.

**fdel**

**fget**

**fset**

**getter** ()

Descriptor to change the getter on a property.

**setter** ()

Descriptor to change the setter on a property.

## 7.1.1 Subpackages

### volatility.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*)

**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 volatility 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: `volatility.cli.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

### volatility.cli.volshell package

**class VolShell**  
Bases: `volatility.cli.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** = 'volatility'

**file\_handler\_class\_factory** (*direct=True*)

**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 volatility 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

### volatility.cli.volshell.generic module

**class NullFileHandler** (*preferred\_name*)

Bases: `_io.BytesIO`, `volatility.framework.interfaces.plugins.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.

**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** (*data*)  
Dummy method

**writelines** (*lines*)  
Dummy method

**class Volshell** (*\*args, \*\*kwargs*)  
Bases: `volatility.framework.interfaces.plugins.PluginInterface`  
Shell environment to directly interact with a memory image.

Args: 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\_layer(layer\_name=None)**

Changes the current default layer

**property config**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**construct\_locals()**

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

**Return type** *List[Tuple[List[str], Any]]*

**property context**

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

**Return type** *ContextInterface*

**create\_configurable(clazz, \*\*kwargs)**

Creates a configurable object, converting arguments to configuration

**property current\_layer****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

**load\_file** (*location=None*, *filename=""*)

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

**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)`

### **volatility.cli.volshell.linux module**

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

Bases: `volatility.cli.volshell.generic.Volshell`

Shell environment to directly interact with a linux memory image.

Args: 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\_layer** (*layer\_name=None*)

Changes the current default layer

**change\_task** (*pid=None*)

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

**property config**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**construct\_locals()**

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

**Return type** `List[Tuple[List[str], Any]]`

**property context**

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

**Return type** `ContextInterface`

**create\_configurable** (*clazz, \*\*kwargs*)

Creates a configurable object, converting arguments to configuration

**property current\_layer**

**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

**list\_tasks** ()

Returns a list of task objects from the primary layer

**load\_file** (*location=None, filename=""*)

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

**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)

### volatility.cli.volshell.mac module

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

Bases: `volatility.cli.volshell.generic.Volshell`

Shell environment to directly interact with a mac memory image.

Args: 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\_layer** (*layer\_name=None*)

Changes the current default layer

**change\_task** (*pid=None*)

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

**property config**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**construct\_locals** ()

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

**Return type** `List[Tuple[List[str], Any]]`



**property context**

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

**Return type** `ContextInterface`

**create\_configurable** (*clazz*, *\*\*kwargs*)

Creates a configurable object, converting arguments to configuration

**property current\_layer****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

**list\_tasks** ()

Returns a list of task objects from the primary layer

**load\_file** (*location=None*, *filename=""*)

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

**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)

## volatility.cli.volshell.windows module

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

Bases: `volatility.cli.volshell.generic.Volshell`

Shell environment to directly interact with a windows memory image.

Args: 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\_layer** (*layer\_name=None*)

Changes the current default layer

**change\_process** (*pid=None*)

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

**property config**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**construct\_locals** ()

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

**Return type** *List[Tuple[List[str], Any]]*

**property context**

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

**Return type** *ContextInterface*

**create\_configurable** (*clazz, \*\*kwargs*)

Creates a configurable object, converting arguments to configuration

**property current\_layer**

**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

**list\_processes** ()

Returns a list of EPROCESS objects from the primary layer

**load\_file** (location=None, filename=)

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

**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

### volatility.cli.text\_renderer module

```
class CLIRenderer(options=None)
    Bases: volatility.framework.interfaces.renderers.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: volatility.cli.text_renderer.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: volatility.cli.text_renderer.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: volatility.cli.text_renderer.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 PrettyTextRenderer (options=None)
    Bases: volatility.cli.text_renderer.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
```

```
class QuickTextRenderer (options=None)
    Bases: volatility.cli.text_renderer.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*

## volatility.cli.volargparse module

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

Bases: *argparse.ArgumentParser*

**add\_argument** (*dest*, ..., *name=value*, ...)

add\_argument(option\_string, option\_string, ..., name=value, ...)

**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: argparse._SubParsersAction
    Class to either select a unique plugin based on a substring, or identify the alternatives.
    add_parser (name, **kwargs)
```

### volatility.framework package

Volatility 3 framework.

```
class subclasses (cls)
    Returns all the (recursive) subclasses of a given class.
    Return type Generator[Type[~T], None, None]
clear_cache (complete=False)
hide_from_subclasses (cls)
    Return type Type
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]]
class noninheritable (value, cls)
    Bases: object
require_interface_version (*args)
    Checks the required version of a plugin.
    Return type None
```



## Subpackages

### volatility.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

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.

---

**Return type** *List[TracebackException]*

## Submodules

### volatility.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: *volatility.framework.interfaces.automagic.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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** = 0

**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]`

## volatility.framework.automagic.linux module

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

Bases: `volatility.framework.automagic.symbol_cache.SymbolBannerCache`

Caches the banners found in the Linux symbol files.

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

**banner\_path** =  `'/home/docs/.cache/volatility3/linux_banners.cache '`

**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`

The `Hierarchical` configuration Dictionary for this `Configurable` object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**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** `load_banners()`

**Return type** `Dict[bytes, List[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`

`os = 'linux'`

`priority = 0`

**classmethod** `save_banners(banners)`

`symbol_name = 'linux_banner'`

**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** `LinuxIntelStacker`

Bases: `volatility.framework.interfaces.automagic.StackerLayerInterface`

**exclusion\_list** = ['mac', 'windows']

**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** = 45

**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: `volatility.framework.automagic.symbol_finder.SymbolFinder`

Linux symbol loader based on uname signature strings.

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

**banner\_cache**

alias of `volatility.framework.automagic.linux.LinuxBannerCache`

**banner\_config\_key** = 'kernel\_banner'

**property** `banners`

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

**Return type** `Dict[bytes, List[str]]`

**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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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 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** = 40

**symbol\_class** = 'volatility.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]`

## volatility.framework.automagic.mac module

**class** `MacBannerCache` (*context, config\_path, \*args, \*\*kwargs*)

Bases: `volatility.framework.automagic.symbol_cache.SymbolBannerCache`

Caches the banners found in the Mac symbol files.

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

**banner\_path** = `'/home/docs/.cache/volatility3/mac_banners.cache'`

**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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**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** `load_banners()`

**Return type** `Dict[bytes, List[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`

`os = 'mac'`

`priority = 0`

**classmethod** `save_banners(banners)`

`symbol_name = 'version'`

**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** `MacIntelStacker`

Bases: `volatility.framework.interfaces.automagic.StackerLayerInterface`

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

**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 = 45`



**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: `volatility.framework.automagic.symbol_finder.SymbolFinder`

Mac symbol loader based on uname signature strings.

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

**banner\_cache**

alias of `volatility.framework.automagic.mac.MacBannerCache`

**banner\_config\_key** = `'kernel_banner'`

**property** `banners`

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

**Return type** `Dict[bytes, List[str]]`

**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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**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 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 = 40`

`symbol_class = 'volatility.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]`

## volatility.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: `volatility.framework.interfaces.automagic.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

#### **property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

#### **property context**

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

**Return type** *ContextInterface*

#### **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\_kernels** – Dictionary containing *GUID*, *age*, *pdb\_name* and *mz\_offset* keys
- **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

#### **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*

**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:`~volatility.framework.interfaces.layers.TranslationLayerRequirement` objects to scan
- **progress\_callback** – Means of providing the user with feedback during long processes

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

`method_module_offset` (*context*, *vlayer*, *progress\_callback=None*)

Return type `Tuple[str, int, Dict[str, Union[bytes, str, int, None]]]`

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

`priority = 30`

`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

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]`

## volatility.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: `volatility.framework.interfaces.automagic.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`

The Hierarchical configuration Dictionary for this Configurable object.

Return type `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

Return type `str`

**property** `context`

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

Return type `ContextInterface`

**create\_stackers\_list**()

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

Return type `List[Type[StackerLayerInterface]]`

**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

**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** (*Optional[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\_stackors** (*plugin*)

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

**Return type** *List[str]*

## volatility.framework.automagic.symbol\_cache module

**class** `SymbolBannerCache` (*context*, *config\_path*, *\*args*, *\*\*kwargs*)

Bases: *volatility.framework.interfaces.automagic.AutomagicInterface*

Runs through all symbols tables and caches their banners.

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

**banner\_path** = *None*

**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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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 load\_banners** ()

**Return type** `Dict[bytes, List[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`

`os = None`

`priority = 0`

```
classmethod save_banners (banners)
```

```
symbol_name = 'banner_name'
```

```
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]`

### volatility.framework.automagic.symbol\_finder module

```
class SymbolFinder (context, config_path)
```

Bases: `volatility.framework.interfaces.automagic.AutomagicInterface`

Symbol loader based on signature strings.

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

```
banner_cache = None
```

```
banner_config_key = 'banner'
```

```
property banners
```

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

**Return type** `Dict[bytes, List[str]]`

```
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
```

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

```
property config_path
```

The configuration path on which this configurable lives.

**Return type** `str`

```
property context
```

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

**Return type** `ContextInterface`

```
find_aslr = 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 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** = 40

**symbol\_class** = 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]*

**volatility.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: *volatility.framework.automagic.windows.DtbSelfReferential*

**second\_pass** (*dtb, data, data\_offset*)

Re-reads over the whole page to validate other records based on the number of pages marked user vs super.

**Parameters**

- **dtb** (*int*) – The identified dtb that needs validating
- **data** (*bytes*) – The chunk of data that contains the dtb to be validated
- **data\_offset** (*int*) – Where, within the layer, the chunk of data lives

**Return type** *Optional[Tuple[int, Any]]*

**Returns** A valid DTB within this page

**class DtbSelfRef64bit**

Bases: *volatility.framework.automagic.windows.DtbSelfReferential*

**second\_pass** (*dtb, data, data\_offset*)

Re-reads over the whole page to validate other records based on the number of pages marked user vs super.

**Parameters**

- **dtb** (*int*) – The identified dtb that needs validating
- **data** (*bytes*) – The chunk of data that contains the dtb to be validated
- **data\_offset** (*int*) – Where, within the layer, the chunk of data lives

**Return type** *Optional[Tuple[int, Any]]*

**Returns** A valid DTB within this page

**class DtbSelfReferential** (*layer\_type, ptr\_struct, ptr\_reference, mask*)

Bases: *volatility.framework.automagic.windows.DtbTest*

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

**second\_pass** (*dtb, data, data\_offset*)

Re-reads over the whole page to validate other records based on the number of pages marked user vs super.

**Parameters**

- **dtb** (`int`) – The identified dtb that needs validating
- **data** (`bytes`) – The chunk of data that contains the dtb to be validated
- **data\_offset** (`int`) – Where, within the layer, the chunk of data lives

**Return type** `Optional[Tuple[int, Any]]`**Returns** A valid DTB within this page**class** `DtbTest` (*layer\_type, ptr\_struct, ptr\_reference, mask*)Bases: `object`

This class generically contains the tests for a page based on a set of class parameters.

When constructed it contains all the information necessary to extract a specific index from a page and determine whether it points back to that page's offset.

**second\_pass** (*dtb, data, data\_offset*)

Re-reads over the whole page to validate other records based on the number of pages marked user vs super.

**Parameters**

- **dtb** (`int`) – The identified dtb that needs validating
- **data** (`bytes`) – The chunk of data that contains the dtb to be validated
- **data\_offset** (`int`) – Where, within the layer, the chunk of data lives

**Return type** `Optional[Tuple[int, Any]]`**Returns** A valid DTB within this page**class** `DtbTest32bit`Bases: `volatility.framework.automagic.windows.DtbTest`**second\_pass** (*dtb, data, data\_offset*)

Re-reads over the whole page to validate other records based on the number of pages marked user vs super.

**Parameters**

- **dtb** (`int`) – The identified dtb that needs validating
- **data** (`bytes`) – The chunk of data that contains the dtb to be validated
- **data\_offset** (`int`) – Where, within the layer, the chunk of data lives

**Return type** `Optional[Tuple[int, Any]]`**Returns** A valid DTB within this page**class** `DtbTest64bit`Bases: `volatility.framework.automagic.windows.DtbTest`**second\_pass** (*dtb, data, data\_offset*)

Re-reads over the whole page to validate other records based on the number of pages marked user vs super.

**Parameters**

- **dtb** (`int`) – The identified dtb that needs validating
- **data** (`bytes`) – The chunk of data that contains the dtb to be validated
- **data\_offset** (`int`) – Where, within the layer, the chunk of data lives

**Return type** `Optional[Tuple[int, Any]]`

**Returns** A valid DTB within this page

**class DtbTestPae**

Bases: *volatility.framework.automagic.windows.DtbTest*

**second\_pass** (*dtb, data, data\_offset*)

PAE top level directory tables contains four entries and the self-referential pointer occurs in the second level of tables (so as not to use up a full quarter of the space). This is very high in the space, and occurs in the fourth (last quarter) second-level table. The second-level tables appear always to come sequentially directly after the real dtb. The value for the real DTB is therefore four page earlier (and the fourth entry should point back to the *dtb* parameter this function was originally passed.

**Parameters**

- **dtb** (*int*) – The identified self-referential pointer that needs validating
- **data** (*bytes*) – The chunk of data that contains the dtb to be validated
- **data\_offset** (*int*) – Where, within the layer, the chunk of data lives

**Return type** *Optional[Tuple[int, Any]]*

**Returns** Returns the actual DTB of the PAE space

**class PageMapScanner** (*tests*)

Bases: *volatility.framework.interfaces.layers.ScannerInterface*

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

**property context**

**Return type** *Optional[ContextInterface]*

**property layer\_name**

**Return type** *Optional[str]*

**overlap** = 16384

**tests** = [*<volatility.framework.automagic.windows.DtbTest64bit object>*, *<volatility.fra*

The default tests to run when searching for DTBs

**thread\_safe** = True

**version** = (0, 0, 0)

**class WinSwapLayers** (*context, config\_path, \*args, \*\*kwargs*)

Bases: *volatility.framework.interfaces.automagic.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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*

**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

**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: `volatility.framework.interfaces.automagic.StackerLayerInterface`

**exclusion\_list** = ['mac', 'linux']

**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

**classmethod stacker\_slow\_warning** ()

### **class WintelHelper** (*context, config\_path, \*args, \*\*kwargs*)

Bases: `volatility.framework.interfaces.automagic.AutomagicInterface`

Windows DTB finder based on self-referential pointers.

This class adheres to the `AutomagicInterface` interface and both determines the directory table base of an intel layer if one hasn't been specified, and constructs the intel layer if necessary (for example when reconstructing a pre-existing configuration).

It will scan for existing TranslationLayers that do not have a DTB using the `PageMapScanner`

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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`



**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** = 20

**tests** = [*<volatility.framework.automatic.windows.DtbTest64bit object>*, *<volatility.fra*

**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]*

## volatility.framework.configuration package

### Submodules

#### volatility.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: `volatility.framework.interfaces.configuration.SimpleTypeRequirement`

A requirement type that contains a boolean value.

##### Parameters

- **name** (`str`) – The name of the requirement
- **description** (`Optional[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**

Returns the default value if one is set.

**Return type** `Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** `str`

**instance\_type**

alias of `bool`

**property name**

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** `str`

**property optional**

Whether the Requirement is optional or not.

**Return type** `bool`

**remove\_requirement** (*requirement*)

Always raises a `TypeError` as instance requirements cannot have children.

**property requirements**

Returns a dictionary of all the child requirements, indexed by name.

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

**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: `volatility.framework.interfaces.configuration.SimpleTypeRequirement`

A requirement type that contains a byte string.

**Parameters**

- **name** (`str`) – The name of the requirement
- **description** (`Optional[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**

Returns the default value if one is set.

**Return type** *Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]*

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** *str*

**instance\_type**

alias of *bytes*

**property name**

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** *str*

**property optional**

Whether the Requirement is optional or not.

**Return type** *bool*

**remove\_requirement** (*requirement*)

Always raises a *TypeError* as instance requirements cannot have children.

**property requirements**

Returns a dictionary of all the child requirements, indexed by name.

**Return type** *Dict[str, RequirementInterface]*

**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: `volatility.framework.interfaces.configuration.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**

Returns the default value if one is set.

**Return type** `Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** `str`

**property name**

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR ('.' by default) since this is used within the configuration hierarchy.

**Return type** `str`

**property optional**

Whether the Requirement is optional or not.

**Return type** `bool`

**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**

Returns a dictionary of all the child requirements, indexed by name.

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

**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: `volatility.framework.configuration.requirements.MultiRequirement`, `volatility.framework.interfaces.configuration.ConfigurableRequirementInterface`

Allows a variable length list of requirements.

**Parameters**

- **name** (*str*) – The name of the requirement
- **description** (*Optional[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**

Returns the default value if one is set.

**Return type** `Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** `str`

**classmethod get\_requirements()**

**Return type** `List[RequirementInterface]`

**property name**

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** `str`

**abstract new\_requirement** (*index*)

Builds a new requirement based on the specified index.

**Return type** `RequirementInterface`

**property optional**

Whether the Requirement is optional or not.

**Return type** `bool`

**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**

Returns a dictionary of all the child requirements, indexed by name.

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

**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: *volatility.framework.interfaces.configuration.SimpleTypeRequirement*

A requirement type that contains a single integer.

**Parameters**

- **name** (*str*) – The name of the requirement
- **description** (*Optional[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**

Returns the default value if one is set.

**Return type** *Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]*

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** *str*

**instance\_type**

alias of *int*

**property name**

The name of the Requirement.



Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** `str`

**property optional**

Whether the Requirement is optional or not.

**Return type** `bool`

**remove\_requirement** (*requirement*)

Always raises a `TypeError` as instance requirements cannot have children.

**property requirements**

Returns a dictionary of all the child requirements, indexed by name.

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

**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: `volatility.framework.configuration.requirements.ComplexListRequirement`

Allows a variable length list of layers that must exist.

**Parameters**

- **name** (*str*) – The name of the requirement
- **description** (*Optional[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**

Returns the default value if one is set.

**Return type** *Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]*

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** *str*

**classmethod get\_requirements** ()

**Return type** *List[RequirementInterface]*

**property name**

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** *str*

**new\_requirement** (*index*)

Constructs a new requirement based on the specified index.

**Return type** *RequirementInterface*

**property optional**

Whether the Requirement is optional or not.

**Return type** *bool*

**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**

Returns a dictionary of all the child requirements, indexed by name.

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

**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: `volatility.framework.interfaces.configuration.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
- **The maximum number of acceptable elements this list can contain** (*max\_elements*;) –
- **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**

Returns the default value if one is set.

**Return type** `Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** `str`

**property name**

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** `str`

**property optional**

Whether the Requirement is optional or not.

**Return type** `bool`

**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**

Returns a dictionary of all the child requirements, indexed by name.

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

**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 MultiRequirement** (*name, description=None, default=None, optional=False*)

Bases: *volatility.framework.interfaces.configuration.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** (*Optional[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**

Returns the default value if one is set.

**Return type** *Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]*

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** *str*

**property name**

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** *str*

**property optional**

Whether the Requirement is optional or not.

**Return type** `bool`

**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**

Returns a dictionary of all the child requirements, indexed by name.

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

**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: `volatility.framework.configuration.requirements.VersionRequirement`

Args: 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]*

#### property default

Returns the default value if one is set.

**Return type** *Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]*

#### property description

A short description of what the Requirement is designed to affect or achieve.

**Return type** *str*

#### property name

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** *str*

#### property optional

Whether the Requirement is optional or not.

**Return type** *bool*

#### 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

Returns a dictionary of all the child requirements, indexed by name.

**Return type** *Dict[str, RequirementInterface]*

#### 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: *volatility.framework.interfaces.configuration.SimpleTypeRequirement*

A requirement type that contains a single unicode string.

**Parameters**

- **name** (*str*) – The name of the requirement
- **description** (*Optional[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**

Returns the default value if one is set.

**Return type** *Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]*

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** *str*

**instance\_type**

alias of *str*



**property name**

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** `str`

**property optional**

Whether the Requirement is optional or not.

**Return type** `bool`

**remove\_requirement** (*requirement*)

Always raises a `TypeError` as instance requirements cannot have children.

**property requirements**

Returns a dictionary of all the child requirements, indexed by name.

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

**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: `volatility.framework.interfaces.configuration.ConstructableRequirementInterface`, `volatility.framework.interfaces.configuration.ConfigurableRequirementInterface`

Class maintaining the limitations on what sort of symbol spaces are acceptable.

Args: 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**

Returns the default value if one is set.

**Return type** *Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]*

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** *str*

**property name**

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** *str*

**property optional**

Whether the Requirement is optional or not.

**Return type** *bool*

**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**

Returns a dictionary of all the child requirements, indexed by name.

**Return type** *Dict[str, RequirementInterface]*

**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*)  
*volatility.framework.interfaces.configuration.ConstructableRequirementInterface, volatility.framework.interfaces.configuration.ConfigurableRequirementInterface*

Bases:

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** (*Optional[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** (*Optional[List]*) – A list of valid operating systems which can satisfy this requirement
- **architectures** (*Optional[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**

Returns the default value if one is set.

**Return type** `Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** `str`

**property name**

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** `str`

**property optional**

Whether the Requirement is optional or not.

**Return type** `bool`

**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**

Returns a dictionary of all the child requirements, indexed by name.

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

**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: `volatility.framework.configuration.requirements.StringRequirement`

A requirement type that contains a single unicode string that is a valid URI.

#### Parameters

- **name** (`str`) – The name of the requirement
- **description** (`Optional[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**

Returns the default value if one is set.

**Return type** `Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** `str`

**instance\_type**

alias of `str`

**property name**

The name of the Requirement.

Names cannot contain `CONFIG_SEPARATOR` ('.' by default) since this is used within the configuration hierarchy.

**Return type** `str`

**property optional**

Whether the Requirement is optional or not.

**Return type** `bool`

**remove\_requirement** (*requirement*)

Always raises a `TypeError` as instance requirements cannot have children.

**property requirements**

Returns a dictionary of all the child requirements, indexed by name.

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

**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: `volatility.framework.interfaces.configuration.RequirementInterface`

Args: *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]`

**property default**

Returns the default value if one is set.

**Return type** `Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** `str`

**property name**

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** `str`

**property optional**

Whether the Requirement is optional or not.

**Return type** `bool`

**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**

Returns a dictionary of all the child requirements, indexed by name.

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

**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

## volatility.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

**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

**LINUX\_BANNERS\_PATH** = '/home/docs/.cache/volatility3/linux\_banners.cache'

"Default location to record information about available linux banners

**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

**MAC\_BANNERS\_PATH** = '/home/docs/.cache/volatility3/mac\_banners.cache'

"Default location to record information about available mac banners

**PACKAGE\_VERSION** = '2.0.0-beta.1'

The canonical version of the volatility package

**PARALLELISM** = <Parallelism.Off: 0>

Default value to the parallelism setting used throughout volatility

**PLUGINS\_PATH** = ['/home/docs/checkouts/readthedocs.org/user\_builds/volatility3/checkouts/lat

Default list of paths to load plugins from (volatility/plugins and volatility/framework/plugins)

**class Parallelism**(*value*)

Bases: `enum.IntEnum`

An enumeration listing the different types of parallelism applied to volatility.

**Multiprocessing** = 2

**Off** = 0

**Threading** = 1



**ProgressCallback**

Type information for ProgressCallback objects

alias of `Optional[Callable[[float, str], None]]`

**SYMBOL\_BASEPATHS** = `['/home/docs/checkouts/readthedocs.org/user_builds/volatility3/checkouts/']`

Default list of paths to load symbols from (volatility/symbols and volatility/framework/symbols)

**Subpackages****volatility.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)

**volatility.framework.constants.windows package**

Volatility 3 Linux 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

**volatility.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 Context**

Bases: `volatility.framework.interfaces.context.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** `volatility.framework.exceptions.LayerException` – if the layer is already present, or has unmet dependencies

**Return type** `None`

**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**

Returns a mutable copy of the configuration, but does not allow the whole configuration to be altered.

**Return type** *HierarchicalDict*

**property layers**

A LayerContainer object, allowing access to all data and translation layers currently available within the context.

**Return type** *LayerContainer*

**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 occupys from offset location within the layer named layer\_name

**Return type** *ModuleInterface*

**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**

The space of all symbols that can be accessed within this context.

**Return type** *SymbolSpaceInterface*

```
class Module(context, module_name, layer_name, offset, symbol_table_name=None, native_layer_name=None)
```

Bases: `volatility.framework.interfaces.context.ModuleInterface`

Constructs a new os-independent module.

#### Parameters

- **context** (`ContextInterface`) – The context within which this module will exist
- **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
- **symbol\_table\_name** (`Optional[str]`) – The name of an associated symbol table
- **native\_layer\_name** (`Optional[str]`) – The default native layer for objects constructed by the module

#### property context

Context that the module uses.

Return type `ContextInterface`

#### get\_enumeration(name)

Returns an enumeration from the module.

Return type `Template`

#### get\_symbol(name)

Returns a symbol from the module.

Return type `SymbolInterface`

#### get\_type(name)

Returns a type from the module.

Return type `Template`

#### has\_enumeration(name)

Determines whether an enumeration is present in the module.

Return type `bool`

#### has\_symbol(name)

Determines whether a symbol is present in the module.

Return type `bool`

#### has\_type(name)

Determines whether a type is present in the module.

Return type `bool`

#### property layer\_name

Layer name in which the Module resides.

Return type `str`

#### property name

The name of the constructed module.

Return type `str`

#### 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** (*Optional[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, \*\*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

**Return type** *ObjectInterface*

**property offset**

Returns the offset that the module resides within the layer of *layer\_name*.

**Return type** *int*

**class ModuleCollection** (*modules*)

Bases: *object*

Class to contain a collection of *SizedModules* and reason about their contents.

**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*

**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]]]*

**property modules**

A name indexed dictionary of modules using that name in this collection.

**Return type** *Dict[str, List[SizedModule]]*

**class SizedModule** (*context, module\_name, layer\_name, offset, size, symbol\_table\_name=None, native\_layer\_name=None*)

Bases: *volatility.framework.contexts.Module*

Constructs a new os-independent module.

**Parameters**

- **context** (*ContextInterface*) – The context within which this module will exist
- **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
- **symbol\_table\_name** (*Optional[str]*) – The name of an associated symbol table
- **native\_layer\_name** (*Optional[str]*) – The default native layer for objects constructed by the module

**property context**

Context that the module uses.

**Return type** *ContextInterface*

**get\_enumeration** (*name*)

Returns an enumeration from the module.

**Return type** *Template*

**get\_symbol** (*name*)

Returns a symbol from the module.

**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.

**Return type** *Template*

**has\_enumeration** (*name*)

Determines whether an enumeration is present in the module.

**Return type** *bool*

**has\_symbol** (*name*)

Determines whether a symbol is present in the module.

**Return type** *bool*

**has\_type** (*name*)

Determines whether a type is present in the module.

**Return type** *bool*

**property hash**

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

**Return type** *str*

**property layer\_name**

Layer name in which the Module resides.

**Return type** *str*

**property name**

The name of the constructed module.

**Return type** `str`

**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** (`Optional[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*, *\*\*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

**Return type** `ObjectInterface`

**property offset**

Returns the offset that the module resides within the layer of `layer_name`.

**Return type** `int`

**property size**

Returns the size of the module (0 for unknown size)

**Return type** `int`

**get\_module\_wrapper** (*method*)

Returns a symbol using the `symbol_table_name` of the Module.

**Return type** `Callable`

## volatility.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

#### volatility.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: *volatility.framework.interfaces.configuration.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** `ContextInterface`

**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** = []

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 call-back 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** ()**volatility.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: *volatility.framework.interfaces.configuration.RequirementInterface*

Requires a specific class.

This is used as means to serialize specific classes for `TranslationLayerRequirement` and `SymbolTableRequirement` classes.

Args: 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**

Contains the actual chosen class based on the configuration value's class name.

**Return type** *Type*

**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**

Returns the default value if one is set.

**Return type** *Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]*

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** *str*

**property name**

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** *str*

**property optional**

Whether the Requirement is optional or not.

**Return type** *bool*

**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**

Returns a dictionary of all the child requirements, indexed by name.

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

**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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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: `volatility.framework.interfaces.configuration.RequirementInterface`

Simple Abstract class to provide build\_required\_config.

**Parameters**

- **name** (`str`) – The name of the requirement
- **description** (`Optional[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**

Returns the default value if one is set.

**Return type** `Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** `str`

**property name**

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** `str`

**property optional**

Whether the Requirement is optional or not.

**Return type** `bool`

**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**

Returns a dictionary of all the child requirements, indexed by name.

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

**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: `volatility.framework.interfaces.configuration.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).

Args: 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**

Returns the default value if one is set.

**Return type** `Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** `str`

**property name**

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** `str`

**property optional**

Whether the Requirement is optional or not.

**Return type** `bool`

**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**

Returns a dictionary of all the child requirements, indexed by name.

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

**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: `collections.abc.Mapping`

The core of configuration data, it is a mapping class that stores keys within itself, and also stores lower hierarchies.

**Parameters**

- **initial\_dict** (`Optional[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**

Returns just the data-containing mappings on this level of the Hierarchy.

**Return type** *Dict*

**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**

Specifies the hierarchy separator in use in this HierarchyDict.

**Return type** *str*

**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** (`Optional[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**

Returns the default value if one is set.

**Return type** `Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** `str`

**property name**

The name of the Requirement.

Names cannot contain CONFIG\_SEPARATOR (‘.’ by default) since this is used within the configuration hierarchy.

**Return type** `str`

**property optional**

Whether the Requirement is optional or not.

**Return type** `bool`

**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**

Returns a dictionary of all the child requirements, indexed by name.

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

**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: `volatility.framework.interfaces.configuration.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** (`Optional[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**

Returns the default value if one is set.

**Return type** `Union[int, bool, bytes, str, List[Union[int, bool, bytes, str]], None]`

**property description**

A short description of what the Requirement is designed to affect or achieve.

**Return type** `str`

**instance\_type**

alias of `bool`

**property name**

The name of the Requirement.

Names cannot contain `CONFIG_SEPARATOR` (`'` by default) since this is used within the configuration hierarchy.

**Return type** `str`

**property optional**

Whether the Requirement is optional or not.

**Return type** `bool`

**remove\_requirement** (*requirement*)

Always raises a `TypeError` as instance requirements cannot have children.

**property requirements**

Returns a dictionary of all the child requirements, indexed by name.

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

**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**

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*

## volatility.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

**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**

Returns the configuration object for this context.

**Return type** *HierarchicalDict*

**abstract property layers**

Returns the memory object for the context.

**Return type** *LayerContainer*

**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 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** (*Optional[str]*) – The layer this object references (should it be a pointer or similar)

**Returns** A fully constructed object

**abstract property symbol\_space**

Returns the symbol\_space for the context.

This object must support the *SymbolSpaceInterface*

**Return type** *SymbolSpaceInterface*

**class ModuleInterface** (*context, module\_name, layer\_name, offset, symbol\_table\_name=None, native\_layer\_name=None*)

Bases: *object*

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
- **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
- **symbol\_table\_name** (*Optional[str]*) – The name of an associated symbol table
- **native\_layer\_name** (*Optional[str]*) – The default native layer for objects constructed by the module

**property context**

Context that the module uses.

**Return type** *ContextInterface*

**get\_enumeration** (*name*)

Returns an enumeration from the module.

**Return type** *Template*

**get\_symbol** (*name*)

Returns a symbol from the module.

**Return type** *SymbolInterface*

**get\_type** (*name*)

Returns a type from the module.

**Return type** *Template*

**has\_enumeration** (*name*)

Determines whether an enumeration is present in the module.

**Return type** *bool*

**has\_symbol** (*name*)

Determines whether a symbol is present in the module.

**Return type** *bool*

**has\_type** (*name*)

Determines whether a type is present in the module.

**Return type** `bool`

**property layer\_name**

Layer name in which the Module resides.

**Return type** `str`

**property name**

The name of the constructed module.

**Return type** `str`

**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** (`Optional[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*, *\*\*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

**Return type** `ObjectInterface`

**Returns** The constructed object

**property offset**

Returns the offset that the module resides within the layer of `layer_name`.

**Return type** `int`

**volatility.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: *volatility.framework.interfaces.configuration.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**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** *int*

**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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**property dependencies**

A list of other layer names required by this layer.

---

**Note:** DataLayers must never define other layers

---

**Return type** *List[str]*

**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`

Returns the maximum valid address of the space.

**Return type** `int`

**property** `metadata`

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**abstract** `property minimum_address`

Returns the minimum valid address of the space.

**Return type** `int`

**property** `name`

Returns the layer name.

**Return type** `str`

**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** (*Optional[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: *collections.abc.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 Layer-Container

**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: `volatility.framework.interfaces.configuration.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**

Return type `Optional[ContextInterface]`

**property layer\_name**

Return type `Optional[str]`

**thread\_safe** = `False`

**version** = `(0, 0, 0)`

**class TranslationLayerInterface** (*context, config\_path, name, metadata=None*)

Bases: `volatility.framework.interfaces.layers.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**

Returns a mask which encapsulates all the active bits of an address for this layer.

Return type `int`

**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**

The Hierarchical configuration Dictionary for this Configurable object.

Return type `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

Return type `str`

**property context**

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

Return type `ContextInterface`

**abstract property dependencies**

Returns a list of layer names that this layer translates onto.

Return type `List[str]`

**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**

Returns the maximum valid address of the space.

**Return type** `int`

**property metadata**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**abstract property minimum\_address**

Returns the minimum valid address of the space.

**Return type** `int`

**property name**

Returns the layer name.

**Return type** `str`

**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** (*Optional[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`

## volatility.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: *volatility.framework.interfaces.objects.ReadOnlyMapping*

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 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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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`.

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**abstract** `write(value)`

Writes the new value into the format at the offset the object currently resides at.

**class** `ReadOnlyMapping(dictionary)`

Bases: `collections.abc.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 at 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.

#### property children

The children of this template (such as member types, sub-types and base-types where they are relevant).

Used to traverse the template tree.

**Return type** `List[Template]`

**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**

Returns the size of the template.

**Return type** `int`

**update\_vol** (*\*\*new\_arguments*)

Updates the keyword arguments with values that will **not** be carried across to clones.

**Return type** `None`

**property vol**

Returns a volatility information object, much like the *ObjectInformation* provides.

**Return type** `ReadOnlyMapping`

**volatility.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: `io.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.

**seek()**

Change stream position.

Change the stream position to the given byte offset. The offset is interpreted relative to the position indicated by whence. Values for whence are:

- 0 – start of stream (the default); offset should be zero or positive
- 1 – current stream position; offset may be negative
- 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: [volatility.framework.interfaces.configuration.ConfigurableInterface](#), [volatility.framework.interfaces.configuration.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

#### **property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

#### **property context**

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

**Return type** *ContextInterface*

#### **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)

## volatility.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.

**property name**

Alias for field number 0

**property type**

Alias for field number 1

```
class ColumnSortKey
```

```
    Bases: object
```

```
    ascending = 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 = (<class 'int'>, <class 'str'>, <class 'float'>, <class 'bytes'>, <class 'None'>)
```

```
    abstract children(node)
```

Returns the subnodes of a particular node in order.

Return type `List[TreeNode]`

```
    abstract property columns
```

Returns the available columns and their ordering and types.

**Return type** `List[Column]`

**abstract is\_ancestor** (*node, descendant*)

Returns true if descendent 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**

Indicates that population has completed and the tree may now be manipulated separately.

**Return type** `bool`

**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, ~_Type], ~_Type]`) – The visitor to apply to the nodes under the initial node
- **initial\_accumulator** (`~_Type`) – An accumulator that allows data to be transferred between one visitor call to the next

- **sort\_key** (`Optional[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: `collections.abc.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`

Returns the parent node of this node or `None`.

**Return type** `Optional[TreeNode]`

**abstract property** `path`

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.

**Return type** `str`

**abstract property** `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`

Return the path depth of the current node.

**Return type** `int`

**abstract property** `values`

Returns the list of values from the particular node, based on column index.

**Return type** `List[Union[Type[int], Type[str], Type[float], Type[bytes], Type[datetime], Type[BaseAbsentValue], Type[Disassembly]]]`

## volatility.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**

Returns an iterator of the Enumeration names.

**Return type** *Iterable[Any]*

**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**

Returns None or a NativeTable for handling space specific native types.

**Return type** *NativeTableInterface*

**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**

Returns an iterator of the Symbol names.

**Return type** *Iterable[str]*

**property types**

Returns an iterator of the Symbol type names.

**Return type** *Iterable[str]*

**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: *volatility.framework.interfaces.symbols.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**

Returns an iterator of the Enumeration names.

**Return type** *Iterable[str]*

**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**

Returns None or a *NativeTable* for handling space specific native types.

**Return type** *NativeTableInterface*

**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**

Returns an iterator of the Symbol names.

**Return type** *Iterable[str]*

**property types**

Returns an iterator of the Symbol type names.

**Return type** *Iterable[str]*

**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**

Returns the relative address of the symbol within the compilation unit.

**Return type** `int`

**property constant\_data**

Returns any constant data associated with the symbol.

**Return type** `Optional[bytes]`

**property name**

Returns the name of the symbol.

**Return type** `str`

**property type**

Returns the type that the symbol represents.

**Return type** `Optional[Template]`

**property type\_name**

Returns the name of the type that the symbol represents.

**Return type** `Optional[str]`

**class SymbolSpaceInterface**

Bases: `collections.abc.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: `volatility.framework.interfaces.symbols.BaseSymbolTableInterface`, `volatility.framework.interfaces.configuration.ConfigurableInterface`, `abc.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**del\_type\_class(name)**

Removes the associated class override for a specific Symbol type.

**Return type** *None*

**property enumerations**

Returns an iterator of the Enumeration names.

**Return type** *Iterable[Any]*

**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**

Returns None or a NativeTable for handling space specific native types.

**Return type** `NativeTableInterface`

**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**

Returns an iterator of the Symbol names.

**Return type** `Iterable[str]`

**property types**

Returns an iterator of the Symbol type names.

**Return type** `Iterable[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]`

## volatility.framework.layers package

### Subpackages

## volatility.framework.layers.codecs package

Codecs used for encoding or decoding data should live here

## volatility.framework.layers.scanners package

```
class BytesScanner(needle)
    Bases: volatility.framework.interfaces.layers.ScannerInterface
    property context
        Return type Optional[ContextInterface]
    property layer_name
        Return type Optional[str]
    thread_safe = True
    version = (0, 0, 0)

class MultiStringScanner(patterns)
    Bases: volatility.framework.interfaces.layers.ScannerInterface
    property context
        Return type Optional[ContextInterface]
    property layer_name
        Return type Optional[str]
    thread_safe = True
    version = (0, 0, 0)

class RegExScanner(pattern, flags=0)
    Bases: volatility.framework.interfaces.layers.ScannerInterface
    property context
        Return type Optional[ContextInterface]
    property layer_name
        Return type Optional[str]
    thread_safe = True
    version = (0, 0, 0)
```



## Submodules

### volatility.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

### volatility.framework.layers.crash module

**class WindowsCrashDump32Layer** (*context, config\_path, name*)

Bases: `volatility.framework.layers.segmented.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

**VALIDDUMP** = 1347245380

**property address\_mask**

Returns a mask which encapsulates all the active bits of an address for this layer.

Return type `int`

**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**

The Hierarchical configuration Dictionary for this Configurable object.

Return type `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**crashdump\_json = 'crash'****property dependencies**

Returns a list of the lower layers that this layer is dependent upon.

**Return type** `List[str]`

**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'****classmethod get\_requirements()**

Returns a list of Requirement objects for this type of layer.

**Return type** `List[RequirementInterface]`

**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**

Returns the maximum valid address of the space.

**Return type** `int`

**property metadata**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**property minimum\_address**

Returns the minimum valid address of the space.

**Return type** `int`

**property name**

Returns the layer name.

**Return type** `str`

**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** (*Optional[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]

**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: *volatility.framework.layers.crash.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**

Returns a mask which encapsulates all the active bits of an address for this layer.

Return type *int*

**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**

The Hierarchical configuration Dictionary for this Configurable object.

Return type *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

Return type *str*

**property context**

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

Return type *ContextInterface*

**crashdump\_json** = 'crash64'

**property dependencies**

Returns a list of the lower layers that this layer is dependent upon.

Return type *List[str]*

**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'

**classmethod get\_requirements()**

Returns a list of Requirement objects for this type of layer.

**Return type** `List[RequirementInterface]`

**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**

Returns the maximum valid address of the space.

**Return type** `int`

**property metadata**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**property minimum\_address**

Returns the minimum valid address of the space.

**Return type** `int`

**property name**

Returns the layer name.

**Return type** `str`

**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** (`Optional[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: `volatility.framework.exceptions.LayerException`

Thrown when an error occurs with the underlying Crash file format.

**args**

**with\_traceback** ()

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**class WindowsCrashDumpStacker**

Bases: `volatility.framework.interfaces.automagic.StackerLayerInterface`

**exclusion\_list** = []

**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 call-back function to indicate progress through a scan (if one is necessary)

**Return type** `Optional[DataLayerInterface]`

**stack\_order** = 11

**classmethod** `stacker_slow_warning()`

### **volatility.framework.layers.elf module**

**class** `Elf64Layer(context, config_path, name)`

Bases: `volatility.framework.layers.segmented.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`

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** `int`

**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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**property** `dependencies`

Returns a list of the lower layers that this layer is dependent upon.

**Return type** `List[str]`

**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**

Returns the maximum valid address of the space.

**Return type** `int`

**property** **metadata**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**property** **minimum\_address**

Returns the minimum valid address of the space.

**Return type** `int`

**property** **name**

Returns the layer name.

**Return type** `str`

**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** (*Optional[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: *volatility.framework.interfaces.automagic.StackerLayerInterface*

**exclusion\_list** = []

**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 call-back function to indicate progress through a scan (if one is necessary)

**Return type** *Optional[DataLayerInterface]*

**stack\_order** = 10

**classmethod stacker\_slow\_warning** ()

**exception ElfFormatException** (*layer\_name, \*args*)

Bases: *volatility.framework.exceptions.LayerException*

Thrown when an error occurs with the underlying ELF file format.

**args**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

## volatility.framework.layers.intel module

**class Intel**(context, config\_path, name, metadata=None)

Bases: *volatility.framework.layers.linear.LinearlyMappedLayer*

Translation Layer for the Intel IA32 memory mapping.

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

**property address\_mask**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** *int*

**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*

**property config**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**property dependencies**

Returns a list of the lower layer names that this layer is dependent upon.

**Return type** *List[str]*

**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, 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**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**minimum\_address** = 0

**property name**

Returns the layer name.

**Return type** `str`

**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** (`Optional[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: `volatility.framework.layers.intel.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
```

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** `int`

```
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`

```
property config
```

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

```
property config_path
```

The configuration path on which this configurable lives.

**Return type** `str`

```
property context
```

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

**Return type** `ContextInterface`

```
property dependencies
```

Returns a list of the lower layer names that this layer is dependent upon.

**Return type** `List[str]`

**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, 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**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**minimum\_address** = 0

**property name**

Returns the layer name.

**Return type** `str`

**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** (*Optional*[*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), ('pa
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: *volatility.framework.layers.intel.Intel*

Class for handling Physical Address Extensions for Intel architectures.

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

**property** **address\_mask**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** *int*

**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*

**property** **config**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**property dependencies**

Returns a list of the lower layer names that this layer is dependent upon.

**Return type** `List[str]`

**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, 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**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**minimum\_address = 0**

**property name**

Returns the layer name.

**Return type** `str`

**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** (*Optional[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

**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: `volatility.framework.layers.intel.WindowsMixin`, `volatility.framework.layers.intel.Intel`

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

**property address\_mask**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** `int`



**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*

**property config**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**property dependencies**

Returns a list of the lower layer names that this layer is dependent upon.

**Return type** *List[str]*

**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, 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**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**minimum\_address** = 0

**property name**

Returns the layer name.

**Return type** `str`

**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** (*Optional[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: `volatility.framework.layers.intel.WindowsMixin`, `volatility.framework.layers.intel.Intel32e`

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

**property** `address_mask`

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** `int`

**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`

**property** `config`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**property** `dependencies`

Returns a list of the lower layer names that this layer is dependent upon.

**Return type** `List[str]`

**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, 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**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** *Mapping*

**minimum\_address** = 0

**property name**

Returns the layer name.

**Return type** *str*

**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** (*Optional[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), ('pa

**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: `volatility.framework.layers.intel.WindowsMixin`, `volatility.framework.layers.intel.IntelPAE`

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

**property address\_mask**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** `int`

**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`

**property config**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**property dependencies**

Returns a list of the lower layer names that this layer is dependent upon.

**Return type** `List[str]`

**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, 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**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**minimum\_address = 0**

**property name**

Returns the layer name.

**Return type** `str`

**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** (*Optional[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
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: *volatility.framework.layers.intel.Intel*

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

**property address\_mask**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** *int*

**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*

**property config**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**property dependencies**

Returns a list of the lower layer names that this layer is dependent upon.

**Return type** `List[str]`

**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, 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**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**minimum\_address = 0**



**property name**

Returns the layer name.

**Return type** `str`

**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** (*Optional[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`

**volatility.framework.layers.lime module****exception LimeFormatException** (*layer\_name, \*args*)Bases: *volatility.framework.exceptions.LayerException*

Thrown when an error occurs with the underlying Lime file format.

**args****with\_traceback** ()

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**class LimeLayer** (*context, config\_path, name*)Bases: *volatility.framework.layers.segmented.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**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** *int***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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***property dependencies**

Returns a list of the lower layers that this layer is dependent upon.

**Return type** *List[str]***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`

Returns the maximum valid address of the space.

**Return type** `int`

**property** `metadata`

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**property** `minimum_address`

Returns the minimum valid address of the space.

**Return type** `int`

**property** `name`

Returns the layer name.

**Return type** `str`

**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** (*Optional*[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: *volatility.framework.interfaces.automagic.StackerLayerInterface*

**exclusion\_list** = []

**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 call-back function to indicate progress through a scan (if one is necessary)

**Return type** *Optional*[*DataLayerInterface*]

**stack\_order** = 10

**classmethod stacker\_slow\_warning** ()

**volatility.framework.layers.linear module****class LinearlyMappedLayer** (*context, config\_path, name, metadata=None*)Bases: *volatility.framework.interfaces.layers.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**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** *int***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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***abstract property dependencies**

Returns a list of layer names that this layer translates onto.

**Return type** *List[str]***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`

Returns the maximum valid address of the space.

**Return type** *int*

**property** `metadata`

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** *Mapping*

**abstract** `property minimum_address`

Returns the minimum valid address of the space.

**Return type** *int*

**property** `name`

Returns the layer name.

**Return type** *str*

**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** (*Optional[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*

## volatility.framework.layers.msf module

**exception PDBFormatException** (*layer\_name, \*args*)

Bases: *volatility.framework.exceptions.LayerException*

Thrown when an error occurs with the underlying MSF file format.

**args**

**with\_traceback** ()

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**class PdbMSFStream** (*context, config\_path, name, metadata=None*)

Bases: *volatility.framework.layers.linear.LinearlyMappedLayer*

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

**property address\_mask**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** *int*

**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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**property dependencies**

Returns a list of layer names that this layer translates onto.

**Return type** `List[str]`

**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**

Returns the maximum valid address of the space.

Return type `int`

**property metadata**

Returns a ReadOnly copy of the metadata published by this layer.

Return type `Mapping`

**property minimum\_address**

Returns the minimum valid address of the space.

Return type `int`

**property name**

Returns the layer name.

Return type `str`

**property pdb\_symbol\_table**

Return type `Optional[str]`

**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** (*Optional[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: `volatility.framework.layers.linear.LinearlyMappedLayer`

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

**property address\_mask**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** `int`

**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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**create\_stream\_from\_pages** (*stream\_name, maximum\_size, pages*)

**Return type** `str`

**property dependencies**

Returns a list of the lower layers that this layer is dependent upon.

**Return type** `List[str]`

**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`

Returns the maximum valid address of the space.

**Return type** `int`**property** `metadata`

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`**property** `minimum_address`

Returns the minimum valid address of the space.

**Return type** `int`**property** `name`

Returns the layer name.

**Return type** `str`**property** `page_size`**property** `pdb_symbol_table`**Return type** `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** (*Optional[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*

## volatility.framework.layers.physical module

**class BufferDataLayer** (*context*, *config\_path*, *name*, *buffer*, *metadata=None*)

Bases: *volatility.framework.interfaces.layers.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**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** *int*

**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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**property dependencies**

A list of other layer names required by this layer.

---

**Note:** DataLayers must never define other layers

---

**Return type** `List[str]`

**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**

Returns the largest available address in the space.

**Return type** `int`

**property metadata**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**property minimum\_address**

Returns the smallest available address in the space.

**Return type** `int`

**property name**

Returns the layer name.

**Return type** `str`

**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** (*Optional[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: `volatility.framework.interfaces.layers.DataLayerInterface`

a DataLayer backed by a file on the filesystem.

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

**property address\_mask**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** `int`

**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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**property dependencies**

A list of other layer names required by this layer.

---

**Note:** DataLayers must never define other layers

---

**Return type** `List[str]`

**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**

Returns the location on which this Layer abstracts.

**Return type** `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 maximum\_address**

Returns the largest available address in the space.

**Return type** *int*

**property metadata**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** *Mapping*

**property minimum\_address**

Returns the smallest available address in the space.

**Return type** *int*

**property name**

Returns the layer name.

**Return type** *str*

**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** (*Optional[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`

## volatility.framework.layers.qemu module

### class QemuStacker

Bases: `volatility.framework.interfaces.automagic.StackerLayerInterface`

**exclusion\_list** = []

**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 call-back function to indicate progress through a scan (if one is necessary)

**Return type** `Optional[DataLayerInterface]`

**stack\_order** = 10

**classmethod stacker\_slow\_warning** ()

**class QemuSuspendLayer** (*context*, *config\_path*, *name*, *metadata=None*)

Bases: `volatility.framework.layers.segmented.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

**QEVm\_SUBSECTION** = 5

**QEVm\_VMDESCRIPTION** = 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**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** `int`

**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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**property dependencies**

Returns a list of the lower layers that this layer is dependent upon.

**Return type** `List[str]`

**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`

**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`

Returns the maximum valid address of the space.

**Return type** *int*

**property** `metadata`

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** *Mapping*

**property** `minimum_address`

Returns the minimum valid address of the space.

**Return type** *int*

**property** `name`

Returns the layer name.

**Return type** *str*

**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** (*Optional[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`

### **volatility.framework.layers.registry module**

**exception** `RegistryFormatException(layer_name, *args)`

Bases: `volatility.framework.exceptions.LayerException`

Thrown when an error occurs with the underlying Registry file format.

**args**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**class** `RegistryHive(context, config_path, name, metadata=None)`

Bases: `volatility.framework.layers.linear.LinearlyMappedLayer`

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

**property** `address_mask`

Return a mask that allows for the volatile bit to be set.

**Return type** `int`

**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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**property dependencies**

Returns a list of layer names that this layer translates onto.

**Return type** `List[str]`

**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**

**Return type** `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**

Returns the maximum valid address of the space.

**Return type** `int`

**property metadata**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**property minimum\_address**

Returns the minimum valid address of the space.

**Return type** `int`

**property name**

Returns the layer name.

**Return type** `str`

**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**

Returns the offset for the root cell in this hive.

**Return type** `int`

**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** (*Optional[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: `volatility.framework.exceptions.LayerException`

Thrown when an index that doesn't exist or can't be found occurs.

**args**

**with\_traceback** ()

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

## volatility.framework.layers.resources module

**class JarHandler**

Bases: `urllib.request.BaseHandler`

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

**class ResourceAccessor** (*progress\_callback=None*, *context=None*)

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 URL's contents should be cached

**Return type** *bool*

**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]*

### **volatility.framework.layers.segmented module**

**class NonLinearlySegmentedLayer** (*context*, *config\_path*, *name*, *metadata*=None)

Bases: *volatility.framework.interfaces.layers.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**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** *int*

**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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*



**property dependencies**

Returns a list of the lower layers that this layer is dependent upon.

**Return type** `List[str]`

**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**

Returns the maximum valid address of the space.

**Return type** `int`

**property metadata**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**property minimum\_address**

Returns the minimum valid address of the space.

**Return type** `int`

**property name**

Returns the layer name.

**Return type** `str`

**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** (*Optional[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: `volatility.framework.layers.segmented.NonLinearlySegmentedLayer`,  
`volatility.framework.layers.linear.LinearlyMappedLayer`

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

**property address\_mask**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** `int`

**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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**property dependencies**

Returns a list of the lower layers that this layer is dependent upon.

**Return type** `List[str]`

**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**

Returns the maximum valid address of the space.

**Return type** `int`

**property metadata**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**property minimum\_address**

Returns the minimum valid address of the space.

**Return type** `int`

**property name**

Returns the layer name.

**Return type** `str`

**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** (*Optional[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`

**volatility.framework.layers.vmware module****exception VmwareFormatException** (*layer\_name, \*args*)Bases: *volatility.framework.exceptions.LayerException*

Thrown when an error occurs with the underlying VMware vmem file format.

**args****with\_traceback** ()

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**class VmwareLayer** (*context, config\_path, name, metadata=None*)Bases: *volatility.framework.layers.segmented.SegmentedLayer*

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

**property address\_mask**

Returns a mask which encapsulates all the active bits of an address for this layer.

**Return type** *int***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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***property dependencies**

Returns a list of the lower layers that this layer is dependent upon.

**Return type** *List[str]***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**

Returns the maximum valid address of the space.

**Return type** `int`

**property metadata**

Returns a ReadOnly copy of the metadata published by this layer.

**Return type** `Mapping`

**property minimum\_address**

Returns the minimum valid address of the space.

**Return type** `int`

**property name**

Returns the layer name.

**Return type** `str`

**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** (*Optional[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: *volatility.framework.interfaces.automagic.StackerLayerInterface*

**exclusion\_list** = []

**classmethod stack** (*context, layer\_name, progress\_callback=None*)

Attempt to stack this based on the starting information.

**Return type** *Optional[DataLayerInterface]*

**stack\_order** = 20

**classmethod stacker\_slow\_warning** ()

## volatility.framework.objects package

**class AggregateType** (*context, type\_name, object\_info, size, members*)

Bases: *volatility.framework.interfaces.objects.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 VolTemplateProxy
    Bases:          volatility.framework.interfaces.objects.ObjectInterface.
                  VolTemplateProxy

    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.
```

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.interfaces.objects.ObjectInterface`, `collections.abc.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: `volatility.framework.interfaces.objects.ObjectInterface`, `VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**property count**

Returns the count dynamically.

**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.

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.interfaces.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**conjugate()**

Returns self, the complex conjugate of any int.

**denominator**

the denominator of a rational number in lowest terms

**from\_bytes** (*byteorder, \*, 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.

**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, byteorder, \*, 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.

**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.

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface, VolTemplateProxy*

**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*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**conjugate()**

Returns self, the complex conjugate of any int.

**denominator**

the denominator of a rational number in lowest terms

**from\_bytes** (*byteorder*, \*, *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.

**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, byteorder, \*, 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.

**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.

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**write** (*value*)

Writes the object into the layer of the context at the current offset.

**Return type** `None`

**class Bytes** (*context, type\_name, object\_info, length=1*)

Bases: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface, VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**center** (*width*[, *fillchar*]) → copy of B

Return B centered in a 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 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) → copy of B

Return a copy of B 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** () → `string`

Create a string of hexadecimal numbers from a bytes object. Example: `b'\xb9\x01\xef'.hex()` -> `'b901ef'`.

**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*]) → copy of B

Return B left justified in a string of length width. Padding is done using the specified fill character (default is a space).

**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.

**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*]) → copy of B

Return B right justified in a string of length width. Padding is done using the specified fill character (default is a space)

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**write** (*value*)

Writes the object into the layer of the context at the current offset.

**Return type** *None*

**zfill** (*width*) → copy of B

Pad a numeric string B with zeros on the left, to fill a field of the specified width. B is never truncated.

**class Char** (*context, type\_name, object\_info, data\_format*)

Bases: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface, VolTemplateProxy*

**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*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**conjugate** ()

Returns self, the complex conjugate of any int.

**denominator**

the denominator of a rational number in lowest terms

**from\_bytes** (*byteorder*, \*, *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.

**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*, *byteorder*, \*, *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.

**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.

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**write** (*value*)

Writes the object into the layer of the context at the current offset.

**Return type** *None*

**class ClassType** (*context, type\_name, object\_info, size, members*)

Bases: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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)

**property 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.

**property length**

Alias for field number 0

**property signed**

Alias for field number 2

**class Enumeration** (*context, type\_name, object\_info, base\_type, choices*)

Bases: *volatility.framework.interfaces.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

**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*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---



**Return type** *ObjectInterface*

#### property choices

**Return type** *Dict[str, int]*

#### **conjugate()**

Returns self, the complex conjugate of any int.

#### **denominator**

the denominator of a rational number in lowest terms

#### property description

Returns the chosen name for the value this object contains.

**Return type** *str*

#### **from\_bytes** (*byteorder*, \*, *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.

**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**  
Returns whether the value for the object is a valid choice

**Return type** `bool`

**lookup** (*value=None*)  
Looks up an individual value and returns the associated name.

**Return type** `str`

**numerator**  
the numerator of a rational number in lowest terms

**real**  
the real part of a complex number

**to\_bytes** (*length, byteorder, \*, 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.

**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.

**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**  
Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface, VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**write** (*value*)

Writes the object into the layer of the context at the current offset.

**Return type** *None*

**class Function** (*context, type\_name, object\_info, \*\*kwargs*)

Bases: *volatility.framework.interfaces.objects.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 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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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.

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**conjugate** ()

Returns self, the complex conjugate of any int.

**denominator**

the denominator of a rational number in lowest terms

**from\_bytes** (*byteorder*, *\**, *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.

**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, byteorder, \*, 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.

**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.

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**write** (*value*)

Writes the object into the layer of the context at the current offset.

**Return type** *None*

**class** **Pointer** (*context, type\_name, object\_info, data\_format, subtype=None*)

Bases: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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, \*, 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.

**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, byteorder, \*, 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.

**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.

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**write** (*value*)

Writes the object into the layer of the context at the current offset.

**Return type** `None`

**class PrimitiveObject** (*context, type\_name, object\_info, data\_format*)

Bases: *volatility.framework.interfaces.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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.

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**write** (`value`)

Writes the object into the layer of the context at the current offset.

**Return type** `None`

**class** `String` (`context`, `type_name`, `object_info`, `max_length=1`, `encoding='utf-8'`, `errors='strict'`)

Bases: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`, `VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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*]])  $\rightarrow$  `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*]])  $\rightarrow$  `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*]])  $\rightarrow$  `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.

Use keyword.iskeyword() to test for reserved identifiers such as “def” and “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** (*x, y=None, z=None, /*)

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.

**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 words in the string, using *sep* as the delimiter string.

**sep** The delimiter according which to split the string. *None* (the default value) means split according to any whitespace, and discard empty strings from the result.

**maxsplit** Maximum number of splits to do. -1 (the default value) means no limit.



Splits are done starting at the end of the string and working 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 words in the string, using *sep* as the delimiter string.

**sep** The delimiter according which to split the string. *None* (the default value) means split according to any whitespace, and discard empty strings from the result.

**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 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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**write** (*value*)

Writes the object into the layer of the context at the current offset.

**Return type** *None*

**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: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*, *VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.interfaces.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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.

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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

### volatility.framework.objects.templates module

**class** **ObjectTemplate** (*object\_class, type\_name, \*\*arguments*)

Bases: *volatility.framework.interfaces.objects.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.

**property** **children**

*~volatility.framework.interfaces.objects.ObjectInterface.VolTemplateProxy*)

**Type** Returns the children of the templated object (see

**Type** class

**Return type** *List[Template]*

**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**

~*volatility.framework.interfaces.objects.ObjectInterface.VolTemplateProxy*)

**Type** Returns the children of the templated object (see

**Type** *class*

**Return type** *int*

**update\_vol** (*\*\*new\_arguments*)

Updates the keyword arguments with values that will **not** be carried across to clones.

**Return type** *None*

**property vol**

Returns a volatility information object, much like the *ObjectInformation* provides.

**Return type** *ReadOnlyMapping*

**class ReferenceTemplate** (*type\_name*, *\*\*arguments*)

Bases: *volatility.framework.interfaces.objects.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.

**property children**

The children of this template (such as member types, sub-types and base-types where they are relevant).

Used to traverse the template tree.

**Return type** *List[Template]*

**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**

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*

**update\_vol** (\*\*new\_arguments)

Updates the keyword arguments with values that will **not** be carried across to clones.

**Return type** *None*

**property vol**

Returns a volatility information object, much like the *ObjectInformation* provides.

**Return type** *ReadOnlyMapping*

## volatility.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.

## volatility.framework.plugins package

All core generic plugins.

These modules should only be imported from volatility.plugins NOT volatility.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

### volatility.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: *volatility.framework.interfaces.renderers.ColumnSortKey*

**ascending** = **True**

**class** **NotApplicableValue**

Bases: *volatility.framework.interfaces.renderers.BaseAbsentValue*

Class that represents values which are empty because they don’t make sense for this node.

**class** **NotAvailableValue**

Bases: *volatility.framework.interfaces.renderers.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: *volatility.framework.interfaces.renderers.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** = (`<class 'int'>`, `<class 'str'>`, `<class 'float'>`, `<class 'bytes'>`, `<class 'Disassembly'>`)

**children** (*node*)

Returns the subnodes of a particular node in order.

**Return type** `List[TreeNode]`

**property columns**

Returns the available columns and their ordering and types.

**Return type** `List[Column]`

**is\_ancestor** (*node*, *descendant*)

Returns true if descendant 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** (`Optional[Callable[[TreeNode, ~_Type], ~_Type]]`) – The visitor to be called on each row of the treegrid
- **initial\_accumulator** (`Optional[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**

Indicates that population has completed and the tree may now be manipulated separately.

**Return type** `bool`

**property row\_count**

Returns the number of rows populated.

**Return type** `int`

**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: `volatility.framework.interfaces.renderers.TreeNode`

Class representing a particular node in a tree grid.

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.

**property** `parent`

Returns the parent node of this node or `None`.

**Return type** `Optional[TreeNode]`

**property** `path`

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.

**Return type** `str`

**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`

Return the path depth of the current node.

**Return type** `int`

**property** `values`

Returns the list of values from the particular node, based on column index.

**Return type** `Sequence[Union[Type[int], Type[str], Type[float], Type[bytes], Type[datetime], Type[BaseAbsentValue], Type[Disassembly]]]`

**class UnparsableValue**

Bases: `volatility.framework.interfaces.renderers.BaseAbsentValue`

Class that represents values which are empty because the data cannot be interpreted correctly.

**class UnreadableValue**

Bases: `volatility.framework.interfaces.renderers.BaseAbsentValue`

Class that represents values which are empty because the data cannot be read.

## Submodules

### `volatility.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]`

**volatility.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.

**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, \*, 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.

**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, byteorder, \*, 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.

**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.

**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.

**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*, \*, *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.

**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*, *byteorder*, \*, *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.

**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.

**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]) → copy of B

Return B centered in a 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 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) → copy of B

Return a copy of B 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**() → string

Create a string of hexadecimal numbers from a bytes object. Example: b'\xb9\x01\xef'.hex() -> 'b901ef'.

**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*]) → copy of B

Return B left justified in a string of length width. Padding is done using the specified fill character (default is a space).

**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.

**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*<sub>[, start[, end]]</sub>) → 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*<sub>[, fillchar]</sub>) → copy of B

Return B right justified in a string of length width. Padding is done using the specified fill character (default is a space)

**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*<sub>[, start[, end]]</sub>) → 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*) → copy of B

Pad a numeric string B with zeros on the left, to fill a field of the specified width. B 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*]) → copy of B

Return B centered in a 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 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*) → copy of B

Return a copy of B 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** () → string

Create a string of hexadecimal numbers from a bytes object. Example: `b'\xb9\x01\xef'.hex()` -> `'b901ef'`.

**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* ]) → copy of B  
Return B left justified in a string of length width. Padding is done using the specified fill character (default is a space).

**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.

**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* ]) → copy of `B`

Return `B` right justified in a string of length `width`. Padding is done using the specified fill character (default is a space)

**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*) → copy of `B`

Pad a numeric string `B` with zeros on the left, to fill a field of the specified width. `B` is never truncated.

## volatility.framework.symbols package

### class SymbolSpace

Bases: `volatility.framework.interfaces.symbols.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: `volatility.framework.objects.templates.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.

**property children**

The children of this template (such as member types, sub-types and base-types where they are relevant).

Used to traverse the template tree.

**Return type** `List[Template]`

**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**

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`

**update\_vol(\*\*new\_arguments)**

Updates the keyword arguments with values that will **not** be carried across to clones.

**Return type** `None`

**property vol**

Returns a volatility information object, much like the *ObjectInformation* provides.

**Return type** `ReadOnlyMapping`

**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*)

Bases: `enum.Enum`

An enumeration.

**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

### volatility.framework.symbols.generic package

**class GenericIntelProcess** (*context, type\_name, object\_info, size, members*)

Bases: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**write** (*value*)

Writes the new value into the format at the offset the object currently resides at.

## volatility.framework.symbols.linux package

**class** **LinuxKernelIntermedSymbols** (*\*args, \*\*kwargs*)

Bases: `volatility.framework.symbols.intermed.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\_shift** – An offset by which to alter all returned symbols for this table
- **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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**classmethod create(context, config\_path, sub\_path, filename, native\_types=None, table\_mapping=None, class\_types=None, symbol\_shift=0, 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 volatility/symbols and volatility/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\_shift** (*int*) – An offset by which to alter all returned symbols for this table
- **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**

Returns None or a NativeTable for handling space specific native types.

Return type `NativeTableInterface`

**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

Bases: `volatility.framework.interfaces.configuration.VersionableInterface`

Class with multiple useful linux functions.

**classmethod files\_descriptors\_for\_process** (context, symbol\_table, task)

**classmethod generate\_kernel\_handler\_info** (context, layer\_name, kernel\_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 lookup\_module\_address** (context, 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)

Return type `str`

**version** = (1, 0, 0)

**classmethod walk\_internal\_list** (vmlinux, struct\_name, list\_member, list\_start)

## Subpackages

### volatility.framework.symbols.linux.extensions package

**class dentry** (*context, type\_name, object\_info, size, members*)

Bases: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*, *VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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`

**path** ()

**Return type** `str`

**property** **vol**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`, `VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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`

**reference\_count** ()

**property** `vol`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.StructType`, `collections.abc.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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`

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`, `VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `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`

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: *volatility.framework.symbols.generic.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**get\_core\_size** ()

**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 value for a given symbol 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`

**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`

**member(*attr*=‘member’)**

Specifically named method for retrieving members.

**Return type** `object`

**property num\_syntab**

**property section\_strtab**

**property section\_syntab**

**property vol**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**get\_mnt\_flags** ()

**get\_mnt\_mountpoint** ()

**get\_mnt\_parent** ()

**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*

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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`

**name\_as\_str** ()

**Return type** `str`

**property** **vol**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**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** ()

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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: *volatility.framework.objects.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

**class VolTemplateProxy**

Bases: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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`

**property** `major`

**Return type** `int`

**member** (`attr='member'`)

Specifically named method for retrieving members.

**Return type** `object`

**property** `minor`

**Return type** `int`

**property** `vol`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: *volatility.framework.symbols.generic.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**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`

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`, `VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**get\_mnt\_mountpoint** ()

**get\_mnt\_parent** ()

**get\_mnt\_root** ()

**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** ()

**member** (*attr='member'*)

Specifically named method for retrieving members.

**Return type** `object`



**property vol**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**extended\_flags** = {1: 'VM\_READ', 2: 'VM\_WRITE', 4: 'VM\_EXEC', 8: 'VM\_SHARED', 16:

**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()`

`member(attr='member')`

Specifically named method for retrieving members.

Return type `object`

`perm_flags = {1: 'r', 2: 'w', 4: 'x'}`

`property vol`

Returns the volatility specific object information.

Return type `ReadOnlyMapping`

`write(value)`

Writes the new value into the format at the offset the object currently resides at.

## Submodules

### volatility.framework.symbols.linux.extensions.bash module

**class hist\_entry** (*context, type\_name, object\_info, size, members*)

Bases: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*, *VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**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`

Returns the volatility specific object information.

Return type `ReadOnlyMapping`

`write(value)`

Writes the new value into the format at the offset the object currently resides at.

## volatility.framework.symbols.linux.extensions.elf module

**class** `elf(context, type_name, object_info, size, members)`

Bases: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**write** (*value*)

Writes the new value into the format at the offset the object currently resides at.

**class elf\_phdr** (*\*args, \*\*kwargs*)

Bases: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

Return type *ObjectInterface*

**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**

Returns the volatility specific object information.

Return type *ReadOnlyMapping*

**write**(value)

Writes the new value into the format at the offset the object currently resides at.

**class elf\_sym**(\*args, \*\*kwargs)

Bases: *volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**write** (*value*)

Writes the new value into the format at the offset the object currently resides at.

## Submodules

### volatility.framework.symbols.linux.bash module

**class BashIntermedSymbols** (*\*args, \*\*kwargs*)

Bases: `volatility.framework.symbols.intermed.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\_shift** – An offset by which to alter all returned symbols for this table
- **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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**classmethod create(context, config\_path, sub\_path, filename, native\_types=None, table\_mapping=None, class\_types=None, symbol\_shift=0, 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 volatility/symbols and volatility/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\_shift** (*int*) – An offset by which to alter all returned symbols for this table
- **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**

Returns None or a NativeTable for handling space specific native types.

**Return type** `NativeTableInterface`

**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]

## volatility.framework.symbols.mac package

**class MacKernelIntermedSymbols** (\*args, \*\*kwargs)

Bases: *volatility.framework.symbols.intermed.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\_shift** – An offset by which to alter all returned symbols for this table
- **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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**classmethod create(context, config\_path, sub\_path, filename, native\_types=None, table\_mapping=None, class\_types=None, symbol\_shift=0, 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 volatility/symbols and volatility/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\_shift** (*int*) – An offset by which to alter all returned symbols for this table
- **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**

Returns None or a NativeTable for handling space specific native types.

**Return type** `NativeTableInterface`

```
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

Bases: *volatility.framework.interfaces.configuration.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)
```

```
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, 2, 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

### volatility.framework.symbols.mac.extensions package

```
class fileglob(context, type_name, object_info, size, members)
    Bases: volatility.framework.objects.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: volatility.framework.interfaces.objects.ObjectInterface,
           VolTemplateProxy
    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.
```

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---



**Return type** *ObjectInterface*

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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`

`sockaddr_dl()`

**property** `vol`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

`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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.symbols.generic.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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**

- – the head of the list (`list_head`) –
- – the name of the embedded list member (`member_name`) –
- – the type of each element in the list (`type_name`) –
- – the maximum amount of elements that will be returned (`max_size`) –

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

`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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**get\_ctltype** ()

Returns the type of the sysctl node

Args: None

**Returns** CTLTYPE\_NODE CTLTYPE\_INT CTLTYPE\_STRING CTLTYPE\_QUAD CTLTYPE\_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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`, `VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**write** (*value*)

Writes the new value into the format at the offset the object currently resides at.

## volatility.framework.symbols.windows package

**class WindowsKernelIntermedSymbols** (*\*args, \*\*kwargs*)

Bases: *volatility.framework.symbols.intermed.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\_shift** – An offset by which to alter all returned symbols for this table
- **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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**classmethod create(context, config\_path, sub\_path, filename, native\_types=None, table\_mapping=None, class\_types=None, symbol\_shift=0, 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 volatility/symbols and volatility/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\_shift** (*int*) – An offset by which to alter all returned symbols for this table
- **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**

Returns None or a NativeTable for handling space specific native types.

**Return type** `NativeTableInterface`

**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

### volatility.framework.symbols.windows.extensions package

**class CONTROL\_AREA** (context, type\_name, object\_info, size, members)

Bases: volatility.framework.objects.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: volatility.framework.interfaces.objects.ObjectInterface.  
VolTemplateProxy

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.StructType`, `volatility.framework.symbols.windows.extensions.pool.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: `volatility.framework.interfaces.objects.ObjectInterface`, `VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**get\_device\_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`

**member** (*attr='member'*)

Specifically named method for retrieving members.

**Return type** `object`

**property** **vol**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: *volatility.framework.objects.StructType, volatility.framework.symbols.windows.extensions.pool.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: *volatility.framework.interfaces.objects.ObjectInterface, VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**get\_driver\_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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.symbols.generic.GenericIntelProcess`, `volatility.framework.symbols.windows.extensions.pool.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: `volatility.framework.interfaces.objects.ObjectInterface`, `VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**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** (`kernel_layer=None`)

Return the EPROCESS that owns this thread.

**Return type** `ObjectInterface`

**property** `vol`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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`  
 Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.StructType`, `volatility.framework.symbols.windows.extensions.pool.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: `volatility.framework.interfaces.objects.ObjectInterface`, `VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.StructType`, `volatility.framework.symbols.windows.extensions.pool.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: `volatility.framework.interfaces.objects.ObjectInterface`, `VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`, `VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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\_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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.StructType`, `collections.abc.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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*

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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: *volatility.framework.symbols.windows.extensions.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**get\_commit\_charge** ()

Get the VAD's commit charge (number of committed pages)

**get\_end** ()

Get the VAD's ending virtual address.

**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\_start** ()

Get the VAD's starting virtual 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\_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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**get\_commit\_charge** ()

Get the VAD's commit charge (number of committed pages)

**get\_end** ()

Get the VAD's ending virtual address.

**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*, *winn\_t\_protections*)

Get the VAD's protection constants as a string.

**get\_right\_child** ()

Get the right child member.

**get\_start** ()

Get the VAD's starting virtual 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\_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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.StructType`, `volatility.framework.symbols.windows.extensions.pool.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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:          volatility.framework.interfaces.objects.ObjectInterface.
                  VolTemplateProxy

    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.
```

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

```
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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*, *VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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**

**Return type** `ObjectInterface`

**class VolTemplateProxy**

Bases: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**write** (`value`)

Writes the new value into the format at the offset the object currently resides at.

## Submodules

### `volatility.framework.symbols.windows.extensions.kdbg` module

**class** `KDDEBUGGER_DATA64` (`context`, `type_name`, `object_info`, `size`, `members`)

Bases: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`, `VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**write** (*value*)

Writes the new value into the format at the offset the object currently resides at.

**volatility.framework.symbols.windows.extensions.network module**

**inet\_ntop** (*address\_family*, *packed\_ip*)

Return type `str`

**volatility.framework.symbols.windows.extensions.pe module**

**class IMAGE\_DOS\_HEADER** (*context*, *type\_name*, *object\_info*, *size*, *members*)

Bases: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

Return type `ObjectInterface`

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**write** (*value*)

Writes the new value into the format at the offset the object currently resides at.

**volatility.framework.symbols.windows.extensions.pool module****class** **ExecutiveObject** (*context, type\_name, object\_info, \*\*kwargs*)Bases: *volatility.framework.interfaces.objects.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** **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** **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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface***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**

Returns the volatility specific object information.

Return type `ReadOnlyMapping`

**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: `volatility.framework.objects.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**

Return type `ObjectInterface`

```
class VolTemplateProxy
    Bases:          volatility.framework.interfaces.objects.ObjectInterface.
                  VolTemplateProxy

    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.
```

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

```
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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**get\_object** (*type\_name*, *use\_top\_down*, *executive=False*, *kernel\_symbol\_table=None*, *native\_layer\_name=None*)

Carve an object or data structure from a kernel pool allocation

**Parameters**

- **type\_name** (`str`) – the data structure type name
- **native\_layer\_name** (`Optional[str]`) – the name of the layer where the data originally lived
- **object\_type** – the object type (executive kernel objects only)
- **kernel\_symbol\_table** (`Optional[str]`) – in case objects of a different symbol table are scanned for

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.symbols.windows.extensions.pool.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**get\_object** (*type\_name*, *use\_top\_down*, *executive=False*, *kernel\_symbol\_table=None*, *native\_layer\_name=None*)

Carve an object or data structure from a kernel pool allocation

**Parameters**

- **type\_name** (*str*) – the data structure type name
- **native\_layer\_name** (*Optional[str]*) – the name of the layer where the data originally lived
- **object\_type** – the object type (executive kernel objects only)
- **kernel\_symbol\_table** (*Optional[str]*) – in case objects of a different symbol table are scanned for

**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*

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**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\_valid()**

**Return type** `bool`

**member(attr='member')**

Specifically named method for retrieving members.

**Return type** `object`

**pool\_type\_lookup = {}**

**property vol**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**write(value)**

Writes the new value into the format at the offset the object currently resides at.

**volatility.framework.symbols.windows.extensions.registry module****class CMHIVE** (*context, type\_name, object\_info, size, members*)Bases: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*, *VolTemplateProxy***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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface***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**

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]`

**property vol**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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:          volatility.framework.interfaces.objects.ObjectInterface.
                    VolTemplateProxy

    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.

```

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

Return type *ObjectInterface*

```

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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`,  
`VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---



**Return type** *ObjectInterface*

**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**

Returns the volatility specific object information.

**Return type** *ReadOnlyMapping*

**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: *volatility.framework.objects.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 VolTemplateProxy**

Bases: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy*

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** *ObjectInterface*

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`, `VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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`

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**write** (*value*)

Writes the new value into the format at the offset the object currently resides at.

**class** `RegKeyFlags` (*value*)

Bases: `enum.IntEnum`

An enumeration.

`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`

**class** `RegValueTypes` (*value*)

Bases: `enum.Enum`

An enumeration.

`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`

**classmethod** `get` (*value*)

An alternative method for using this enum when the value may be unknown.

This is used to support unknown value requests in Python <3.6.

**volatility.framework.symbols.windows.extensions.services module****class SERVICE\_HEADER** (*context, type\_name, object\_info, size, members*)Bases: *volatility.framework.objects.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: *volatility.framework.interfaces.objects.ObjectInterface*,  
*VolTemplateProxy***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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**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 structure is valid.

**Return type** `bool`

**member** (*attr*='member')

Specifically named method for retrieving members.

**Return type** `object`

**property vol**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**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: `volatility.framework.objects.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: `volatility.framework.interfaces.objects.ObjectInterface`, `VolTemplateProxy`

**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.

---

**Note:** If new type name does not include a symbol table, the symbol table for the current object is used

---

**Return type** `ObjectInterface`

**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**

Returns the volatility specific object information.

**Return type** `ReadOnlyMapping`

**write** (`value`)

Writes the new value into the format at the offset the object currently resides at.

## Submodules

### `volatility.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()**

**class PdbRetreiver**

Bases: `object`

**retreive\_pdb** (*guid, file\_name, progress\_callback=None*)

**Return type** `Optional[str]`

## volatility.framework.symbols.windows.pdbutil module

**class PDBUtility**

Bases: `object`

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 (downlading if necessary) a windows symbol table

**classmethod** `pdbname_scan` (*ctx, layer\_name, page\_size, pdb\_names, progress\_callback=None, start=None, end=None*)

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)

---

**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

**Return type** `Generator[Dict[str, Union[bytes, str, int, None]], None, None]`

**classmethod** `symbol_table_from_offset` (*context, layer\_name, offset, symbol\_table\_class='volatility.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** (*Optional[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

**class** `PdbSignatureScanner` (*pdb\_names*)

Bases: *volatility.framework.interfaces.layers.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 str's will not match against the data scanned

---

**property** `context`

**Return type** `Optional[ContextInterface]`

**property** `layer_name`

**Return type** `Optional[str]`

**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)
```

## volatility.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 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

### volatility.framework.symbols.intermed module

```
class ISFormatTable(context, config_path, name, json_object, native_types=None, table_mapping=None)
```

Bases: `volatility.framework.interfaces.symbols.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** (*Optional[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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**del\_type\_class** (*name*)

Removes the associated class override for a specific Symbol type.

**Return type** *None*

**property enumerations**

Returns an iterator of the Enumeration names.

**Return type** *Iterable[Any]*

**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**

Returns a metadata object containing information about the symbol table.

**Return type** `Optional[MetadataInterface]`

**property natives**

Returns None or a NativeTable for handling space specific native types.

**Return type** `NativeTableInterface`

**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**

Returns an iterator of the Symbol names.

**Return type** `Iterable[str]`

**property types**

Returns an iterator of the Symbol type names.

**Return type** `Iterable[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]`

**version** = (0, 0, 0)

**class IntermediateSymbolTable** (*context, config\_path, name, isf\_url, native\_types=None, table\_mapping=None, validate=True, class\_types=None, symbol\_shift=0, symbol\_mask=0*)

Bases: `volatility.framework.interfaces.symbols.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 volatility/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** (`Optional[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\_shift** (*int*) – An offset by which to alter all returned symbols for this table
- **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**

The *Hierarchical* configuration *Dictionary* for this *Configurable* object.

**Return type** *HierarchicalDict*

#### **property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

#### **property context**

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

**Return type** *ContextInterface*

#### **classmethod create(context, config\_path, sub\_path, filename, native\_types=None, table\_mapping=None, class\_types=None, symbol\_shift=0, 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 *volatility/symbols* and *volatility/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\_shift** (*int*) – An offset by which to alter all returned symbols for this table

- **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**

Returns None or a NativeTable for handling space specific native types.

**Return type** `NativeTableInterface`

**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: `volatility.framework.symbols.intermed.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** (`Optional[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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**del\_type\_class(name)**

Removes the associated class override for a specific Symbol type.

**Return type** *None*

**property enumerations**

Returns an iterator of the available enumerations.

**Return type** *Iterable[str]*

**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**

Returns a metadata object containing information about the symbol table.

**Return type** `Optional[MetadataInterface]`

**property natives**

Returns None or a NativeTable for handling space specific native types.

**Return type** `NativeTableInterface`

**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**

Returns an iterator of the symbol names.

**Return type** `Iterable[str]`

**property types**

Returns an iterator of the symbol type names.

**Return type** `Iterable[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]`

**version** = (0, 0, 1)

**class** `Version2Format(context, config_path, name, json_object, native_types=None, table_mapping=None)`

Bases: `volatility.framework.symbols.intermed.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** (`Optional[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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** *ContextInterface*

**del\_type\_class** (*name*)

Removes the associated class override for a specific Symbol type.

**Return type** *None*

**property enumerations**

Returns an iterator of the available enumerations.

**Return type** *Iterable[str]*

**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**

Returns a metadata object containing information about the symbol table.

**Return type** `Optional[MetadataInterface]`

**property natives**

Returns None or a NativeTable for handling space specific native types.

**Return type** `NativeTableInterface`

**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**

Returns an iterator of the symbol names.

**Return type** `Iterable[str]`

**property types**

Returns an iterator of the symbol type names.

**Return type** `Iterable[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]`

**version** = (2, 0, 0)

**class Version3Format** (*context*, *config\_path*, *name*, *json\_object*, *native\_types*=None, *table\_mapping*=None)

Bases: `volatility.framework.symbols.intermed.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** (*Optional[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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**del\_type\_class** (*name*)

Removes the associated class override for a specific Symbol type.

**Return type** *None*

**property enumerations**

Returns an iterator of the available enumerations.

**Return type** *Iterable[str]*

**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**

Returns a metadata object containing information about the symbol table.

**Return type** `Optional[MetadataInterface]`

**property natives**

Returns None or a NativeTable for handling space specific native types.

**Return type** `NativeTableInterface`

**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**

Returns an iterator of the symbol names.

**Return type** `Iterable[str]`

**property types**

Returns an iterator of the symbol type names.

**Return type** `Iterable[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]`

**version** = (2, 1, 0)

**class Version4Format** (*context, config\_path, name, json\_object, native\_types=None, table\_mapping=None*)

Bases: `volatility.framework.symbols.intermed.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** (`Optional[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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**del\_type\_class** (*name*)

Removes the associated class override for a specific Symbol type.

**Return type** *None*

**property enumerations**

Returns an iterator of the available enumerations.

**Return type** *Iterable[str]*

**format\_mapping** = {'bool': <class 'volatility.framework.objects.Boolean'>, 'char': <c

**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`

Returns a metadata object containing information about the symbol table.

**Return type** *Optional[MetadataInterface]*

**property** `natives`

Returns None or a NativeTable for handling space specific native types.

**Return type** *NativeTableInterface*

**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`

Returns an iterator of the symbol names.

**Return type** *Iterable[str]*

**property** `types`

Returns an iterator of the symbol type names.

**Return type** *Iterable[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]*

**version** = (4, 0, 0)

```
class Version5Format (context, config_path, name, json_object, native_types=None, table_mapping=None)
```

Bases: `volatility.framework.symbols.intermed.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** (`Optional[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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**del\_type\_class(name)**

Removes the associated class override for a specific Symbol type.

**Return type** `None`

**property enumerations**

Returns an iterator of the available enumerations.

**Return type** `Iterable[str]`

```
format_mapping = {'bool': <class 'volatility.framework.objects.Boolean'>, 'char': <c
```

**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**

Returns a metadata object containing information about the symbol table.

**Return type** *Optional[MetadataInterface]*

**property natives**

Returns None or a NativeTable for handling space specific native types.

Return type `NativeTableInterface`

**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**

Returns an iterator of the symbol names.

Return type `Iterable[str]`

**property types**

Returns an iterator of the symbol type names.

Return type `Iterable[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]`

**version** = (4, 1, 0)

**class Version6Format** (*context*, *config\_path*, *name*, *json\_object*, *native\_types*=None, *table\_mapping*=None)

Bases: `volatility.framework.symbols.intermed.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** (`Optional[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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**del\_type\_class(name)**

Removes the associated class override for a specific Symbol type.

**Return type** *None*

**property enumerations**

Returns an iterator of the available enumerations.

**Return type** *Iterable[str]*

**format\_mapping** = {'bool': <class 'volatility.framework.objects.Boolean'>, 'char': <c

**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**

Returns a MetadataInterface object.

**Return type** `Optional[MetadataInterface]`

**property natives**

Returns None or a NativeTable for handling space specific native types.

**Return type** `NativeTableInterface`

**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**

Returns an iterator of the symbol names.

**Return type** `Iterable[str]`

**property types**

Returns an iterator of the symbol type names.

**Return type** `Iterable[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]`

**version** = (6, 0, 0)

**class** `Version7Format(context, config_path, name, json_object, native_types=None, table_mapping=None)`

Bases: `volatility.framework.symbols.intermed.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** (`Optional[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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**del\_type\_class** (*name*)

Removes the associated class override for a specific Symbol type.

**Return type** `None`

**property enumerations**

Returns an iterator of the available enumerations.

**Return type** `Iterable[str]`

**format\_mapping** = {'bool': <class 'volatility.framework.objects.Boolean'>, 'char': <c

**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**

Returns a MetadataInterface object.

**Return type** `Optional[MetadataInterface]`

#### **property natives**

Returns None or a NativeTable for handling space specific native types.

**Return type** `NativeTableInterface`

#### **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**

Returns an iterator of the symbol names.

**Return type** `Iterable[str]`

#### **property types**

Returns an iterator of the symbol type names.

**Return type** `Iterable[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]`

**version** = (6, 1, 0)

**class Version8Format** (*context*, *config\_path*, *name*, *json\_object*, *native\_types=None*, *table\_mapping=None*)

Bases: `volatility.framework.symbols.intermed.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** (*Optional[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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**del\_type\_class** (*name*)

Removes the associated class override for a specific Symbol type.

**Return type** *None*

**property enumerations**

Returns an iterator of the available enumerations.

**Return type** *Iterable[str]*

**format\_mapping** = {'bool': <class 'volatility.framework.objects.Boolean'>, 'char': <c

**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**

Returns a MetadataInterface object.

**Return type** *Optional[MetadataInterface]*

**property natives**

Returns None or a NativeTable for handling space specific native types.

**Return type** *NativeTableInterface*

**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**

Returns an iterator of the symbol names.

**Return type** *Iterable[str]*

**property types**

Returns an iterator of the symbol type names.

**Return type** *Iterable[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]*

**version** = (6, 2, 0)

## volatility.framework.symbols.metadata module

**class LinuxMetadata** (*json\_data*)

Bases: *volatility.framework.interfaces.symbols.MetadataInterface*

Class to handle the etadata from a Linux symbol table.

Constructor that accepts json\_data.

**class WindowsMetadata** (*json\_data*)

Bases: *volatility.framework.interfaces.symbols.MetadataInterface*

Class to handle the metadata from a Windows symbol table.

Constructor that accepts json\_data.

**property pdb\_age**

**Return type** *Optional[int]*

**property pdb\_guid**

**Return type** *Optional[str]*

**property pe\_version**

**Return type** *Optional[Tuple]*

**property pe\_version\_string**

**Return type** *Optional[str]*



**volatility.framework.symbols.native module****class NativeTable** (*name, native\_dictionary*)Bases: *volatility.framework.interfaces.symbols.NativeTableInterface*

Symbol List that handles Native types.

Args: name: 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**

Returns an iterator of the Enumeration names.

**Return type** *Iterable[str]***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**

Returns None or a NativeTable for handling space specific native types.

**Return type** `NativeTableInterface`

**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**

Returns an iterator of the Symbol names.

**Return type** `Iterable[str]`

**property types**

Returns an iterator of the symbol type names.

**Return type** `Iterable[str]`

**volatility.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**

**Return type** `ReadOnlyMapping`

**Submodules****volatility.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: `volatility.framework.exceptions.LayerException`

Thrown when an address is not valid in the layer it was requested.

**args****with\_traceback** ()

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception LayerException** (*layer\_name*, \*args)

Bases: *volatility.framework.exceptions.VolatilityException*

Thrown when an error occurs dealing with memory and layers.

**args**

**with\_traceback** ()

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception MissingModuleException** (*module*, \*args, \*\*kwargs)

Bases: *volatility.framework.exceptions.VolatilityException*

**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: *volatility.framework.exceptions.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

**args**

**with\_traceback** ()

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception PluginRequirementException**

Bases: *volatility.framework.exceptions.VolatilityException*

Class to allow plugins to indicate that a requirement has not been fulfilled.

**args**

**with\_traceback** ()

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception PluginVersionException**

Bases: *volatility.framework.exceptions.VolatilityException*

Class to allow determining that a required plugin has an invalid version.

**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: *volatility.framework.exceptions.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.

**args**

**with\_traceback** ()

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception SymbolError** (*symbol\_name, table\_name, \*args*)

Bases: *volatility.framework.exceptions.VolatilityException*

Thrown when a symbol lookup has failed.

**args**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception SymbolSpaceError**

Bases: *volatility.framework.exceptions.VolatilityException*

Thrown when an error occurs dealing with Symbolspaces and SymbolTables.

**args**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception UnsatisfiedException** (*unsatisfied*)

Bases: *volatility.framework.exceptions.VolatilityException*

**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.

**args**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

## volatility.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

### volatility.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

### volatility.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: *volatility.framework.interfaces.plugins.PluginInterface*, *volatility.plugins.timeliner.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)

## volatility.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: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)

### **volatility.plugins.linux.check\_creds module**

**class** `Check_creds(context, config_path, progress_callback=None)`

Bases: `volatility.framework.interfaces.plugins.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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**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)

## volatility.plugins.linux.check\_idt module

**class Check\_idt(context, config\_path, progress\_callback=None)**

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)

## volatility.plugins.linux.check\_modules module

**class** `Check_modules(context, config_path, progress_callback=None)`

Bases: `volatility.framework.interfaces.plugins.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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**get\_kset\_modules** (`vmlinux`)

**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)

## volatility.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: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)

### volatility.plugins.linux.elfs module

A module containing a collection of plugins that produce data typically found in Linux's /proc file system.

**class Elf** (*context, config\_path, progress\_callback=None*)

Bases: `volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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)

**volatility.plugins.linux.keyboard\_notifiers module****class Keyboard\_notifiers** (*context, config\_path, progress\_callback=None*)Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***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)**

## volatility.plugins.linux.lsmmod module

A module containing a collection of plugins that produce data typically found in Linux's /proc file system.

**class Lsmmod(context, config\_path, progress\_callback=None)**

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**classmethod get\_requirements()**

Returns a list of Requirement objects for this plugin.

**Return type** `List[RequirementInterface]`

**classmethod list\_modules(context, layer\_name, vmlinux\_symbols)**

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
- **vmlinux\_symbols** (`str`) – The name of the table containing the kernel symbols

**Yields** The modules present in the *layer\_name* layer’s modules list

This function will throw a SymbolError exception if kernel module support is not enabled.

**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** = (1, 0, 0)

## volatility.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: `volatility.framework.interfaces.plugins.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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**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)

## volatility.plugins.linux.malfind module

**class** `Malfind(context, config_path, progress_callback=None)`

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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)

### volatility.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: `volatility.framework.interfaces.plugins.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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**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)

## volatility.plugins.linux.pslist module

**class** `PsList(context, config_path, progress_callback=None)`

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**classmethod create\_pid\_filter** (`pid_list=None`)

Constructs a filter function for process IDs.

**Parameters** **pid\_list** (`Optional[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 list\_tasks** (`context`, `layer_name`, `vmlinux_symbols`, `filter_func=<function PsList.<lambda>>`)

Lists all the tasks 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
- **vmlinux\_symbols** (`str`) – The name of the table containing the kernel symbols

**Yields** Process 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** = (1, 0, 0)

### **volatility.plugins.linux.pstree module**

#### **class PsTree(\*args, \*\*kwargs)**

Bases: *volatility.plugins.linux.pslist.PsList*

Plugin for listing processes in a tree based on their parent process ID.

Args: 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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**classmethod create\_pid\_filter** (*pid\_list=None*)

Constructs a filter function for process IDs.

**Parameters** **pid\_list** (*Optional[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

**find\_level** (*pid*)

Finds how deep the pid is in the processes list.

**classmethod get\_requirements** ()

Returns a list of Requirement objects for this plugin.

**Return type** *List[RequirementInterface]*

**classmethod list\_tasks** (*context, layer\_name, vmlinux\_symbols, filter\_func=<function PsList.<lambda>>>*)

Lists all the tasks 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
- **vmlinux\_symbols** (*str*) – The name of the table containing the kernel symbols

**Yields** Process 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** = (1, 0, 0)

**volatility.plugins.linux.tty\_check module****class tty\_check(context, config\_path, progress\_callback=None)**

Bases: `volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)
```

## volatility.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

### volatility.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: *volatility.framework.interfaces.plugins.PluginInterface, volatility.plugins.timeliner.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)**

**volatility.plugins.mac.check\_syscall module****class Check\_syscall** (*context, config\_path, progress\_callback=None*)Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***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)**

## volatility.plugins.mac.check\_sysctl module

**class Check\_sysctl(context, config\_path, progress\_callback=None)**

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*



**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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)

**volatility.plugins.mac.check\_trap\_table module****class** `Check_trap_table` (*context, config\_path, progress\_callback=None*)Bases: `volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`**property context**

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

**Return type** `ContextInterface`**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)**

## volatility.plugins.mac.ifconfig module

**class Ifconfig(context, config\_path, progress\_callback=None)**

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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)

**volatility.plugins.mac.kauth\_listeners module****class Kauth\_listeners** (*context, config\_path, progress\_callback=None*)Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***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)**

## volatility.plugins.mac.kauth\_scopes module

**class Kauth\_scopes(context, config\_path, progress\_callback=None)**

Bases: `volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**classmethod get\_requirements()**

Returns a list of Requirement objects for this plugin.

**classmethod list\_kauth\_scopes** (*context, layer\_name, darwin\_symbols, filter\_func=<function Kauth\_scopes.<lambda>>>*)

Enumerates the registered kauth scopes and yields each object Uses smear-safe enumeration API

**Return type** `Iterable[Tuple[ObjectInterface, ObjectInterface, 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)`

### `volatility.plugins.mac.kevents` module

**class** `Kevents` (*context, config\_path, progress\_callback=None*)

Bases: `volatility.framework.interfaces.plugins.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_`

`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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

`event_types = {1: 'EVFILT_READ', 2: 'EVFILT_WRITE', 3: 'EVFILT_AIO', 4: 'EVFILT_VN`

**classmethod** `get_requirements()`

Returns a list of Requirement objects for this plugin.

**classmethod** `list_kernel_events` (*context, layer\_name, darwin\_symbols, fil-  
ter\_func=<function Kevents.<lambda>>>*)

Returns the kernel event filters registered

**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



**Return type** `Iterable[Tuple[ObjectInterface, ObjectInterface, 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

`proc_filters = [('NOTE_EXIT', 2147483648), ('NOTE_EXITSTATUS', 67108864), ('NOTE_FORK'`

`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), ('NO`

**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)`

`vnode_filters = [('NOTE_DELETE', 1), ('NOTE_WRITE', 2), ('NOTE_EXTEND', 4), ('NOTE_ATT`

**volatility.plugins.mac.list\_files module****class List\_Files** (*context, config\_path, progress\_callback=None*)Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***classmethod get\_requirements** ()

Returns a list of Requirement objects for this plugin.

**classmethod list\_files** (*context, layer\_name, darwin\_symbols*)**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)

**volatility.plugins.mac.lsmodule module**

A module containing a collection of plugins that produce data typically found in Mac's lsmodule command.

**class Lsmodule(context, config\_path, progress\_callback=None)**

Bases: `volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**classmethod get\_requirements()**

Returns a list of Requirement objects for this plugin.

**classmethod list\_modules(context, layer\_name, darwin\_symbols)**

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
- **darwin\_symbols** (*str*) – 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** = (1, 0, 0)

## volatility.plugins.mac.Isof module

**class** `Isof(context, config_path, progress_callback=None)`

Bases: `volatility.framework.interfaces.plugins.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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**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)

**volatility.plugins.mac.malfind module****class Malfind(context, config\_path, progress\_callback=None)**

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)

### **volatility.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: `volatility.framework.interfaces.plugins.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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**classmethod** `get_requirements()`

Returns a list of Requirement objects for this plugin.

**classmethod** `list_mounts(context, layer_name, darwin_symbols)`

Lists all the mount structures 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
- **darwin\_symbols** (*str*) – 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** = (1, 0, 0)

**volatility.plugins.mac.netstat module****class Netstat** (*context, config\_path, progress\_callback=None*)Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***classmethod get\_requirements** ()

Returns a list of Requirement objects for this plugin.

**classmethod list\_sockets** (*context, layer\_name, darwin\_symbols, filter\_func=<function Netstat.<lambda>>*)

Returns the open socket descriptors of a process

**Return values:****A tuple of 3 elements:**

- 1) The name of the process that opened the socket
- 2) The process ID of the processed that opened the socket
- 3) The address of the associated socket structure

**Return type** *Iterable[Tuple[ObjectInterface, ObjectInterface, 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)

**volatility.plugins.mac.proc\_maps module****class Maps(context, config\_path, progress\_callback=None)**

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)

## volatility.plugins.mac.psaux module

In-memory artifacts from OSX systems.

**class** `Psaux(context, config_path, progress_callback=None)`

Bases: `volatility.framework.interfaces.plugins.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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**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)

**volatility.plugins.mac.pslist module**

**class PsList** (*context, config\_path, progress\_callback=None*)

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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, 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, layer\_name, darwin\_symbols, 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

- **layer\_name** (*str*) – The name of the layer on which to operate
- **darwin\_symbols** (*str*) – The name of the table containing the kernel symbols
- **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*, *layer\_name*, *darwin\_symbols*, *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
- **layer\_name** (*str*) – The name of the layer on which to operate
- **darwin\_symbols** (*str*) – The name of the table containing the kernel symbols
- **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*, *layer\_name*, *darwin\_symbols*, *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
- **layer\_name** (*str*) – The name of the layer on which to operate
- **darwin\_symbols** (*str*) – The name of the table containing the kernel symbols
- **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*, *layer\_name*, *darwin\_symbols*, *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
- **layer\_name** (*str*) – The name of the layer on which to operate
- **darwin\_symbols** (*str*) – The name of the table containing the kernel symbols
- **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*, *layer\_name*, *darwin\_symbols*, *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
- **layer\_name** (*str*) – The name of the layer on which to operate
- **darwin\_symbols** (*str*) – The name of the table containing the kernel symbols
- **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** = (2, 0, 0)

### volatility.plugins.mac.pstree module

**class PsTree** (\*args, \*\*kwargs)

Bases: `volatility.framework.interfaces.plugins.PluginInterface`

Plugin for listing processes in a tree based on their parent process ID.

Args: 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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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)

**volatility.plugins.mac.socket\_filters module****class Socket\_filters(context, config\_path, progress\_callback=None)**

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)
```

## volatility.plugins.mac.timers module

**class Timers** (*context, config\_path, progress\_callback=None*)

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)

**volatility.plugins.mac.trustedbsd module****class Trustedbsd(context, config\_path, progress\_callback=None)**

Bases: `volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)
```

### volatility.plugins.mac.vfsevents module

**class VFSevents** (*context, config\_path, progress\_callback=None*)

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**event\_types** = ['CREATE\_FILE', 'DELETE', 'STAT\_CHANGED', 'RENAME', 'CONTENT\_MODIFIED',

**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)

**volatility.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****volatility.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

### volatility.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`

Return type `Optional[int]`

**class** `HiveList` (*context, config\_path, progress\_callback=None*)

Bases: `volatility.framework.interfaces.plugins.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`

The Hierarchical configuration Dictionary for this Configurable object.

Return type `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

Return type `str`

**property** `context`

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

Return type `ContextInterface`

**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** (*Optional[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)

### volatility.plugins.windows.registry.hivescan module

**class HiveScan** (*context, config\_path, progress\_callback=None*)

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)
```

### volatility.plugins.windows.registry.printkey module

```
class PrintKey(context, config_path, progress_callback=None)
```

Bases: *volatility.framework.interfaces.plugins.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
```

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

```
property config_path
```

The configuration path on which this configurable lives.

**Return type** *str*

```
property context
```

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

**Return type** *ContextInterface*

```
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** (*Optional[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)

**volatility.plugins.windows.registry.userassist module****class UserAssist** (\*args, \*\*kwargs)Bases: *volatility.framework.interfaces.plugins.PluginInterface*

Print userassist registry keys and information.

Args: 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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***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

### volatility.plugins.windows.bigpools module

**class BigPools(context, config\_path, progress\_callback=None)**

Bases: `volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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)**

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, 0, 0)

## volatility.plugins.windows.cachedump module

## volatility.plugins.windows.callbacks module

## volatility.plugins.windows.cmdline module

**class** `CmdLine(context, config_path, progress_callback=None)`

Bases: `volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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)

**volatility.plugins.windows.dlllist module**

**class** `DllList` (*context*, *config\_path*, *progress\_callback*=None)

Bases: `volatility.framework.interfaces.plugins.PluginInterface`, `volatility.plugins.timeliner.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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** (*Optional*[*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)**

**volatility.plugins.windows.driverirp module**

**class DriverIrp** (*context, config\_path, progress\_callback=None*)

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)**

## volatility.plugins.windows.driverscan module

**class DriverScan(context, config\_path, progress\_callback=None)**

Bases: `volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`



**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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)

### **volatility.plugins.windows.dumpfiles module**

**class** `DumpFiles(context, config_path, progress_callback=None)`

Bases: `volatility.framework.interfaces.plugins.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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**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\_object** – the FILE\_OBJECT

**Return type** *Tuple*

**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)

### volatility.plugins.windows.envvars module

**class Envvars** (*context, config\_path, progress\_callback=None*)

Bases: `volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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)

**volatility.plugins.windows.filescan module****class FileScan** (*context, config\_path, progress\_callback=None*)Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***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)

## volatility.plugins.windows.getservicesids module

**class GetServiceSIDs** (*\*args, \*\*kwargs*)

Bases: *volatility.framework.interfaces.plugins.PluginInterface*

Lists process token sids.

Args: 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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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`

## volatility.plugins.windows.getsids module

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

Bases: `volatility.framework.interfaces.plugins.PluginInterface`

Print the SIDs owning each process

Args: 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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**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]*

**volatility.plugins.windows.handles module****class Handles(\*args, \*\*kwargs)**

Bases: *volatility.framework.interfaces.plugins.PluginInterface*

Lists process open handles.

Args: 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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)

**volatility.plugins.windows.hashdump module****volatility.plugins.windows.info module****class Info(context, config\_path, progress\_callback=None)**

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)

**volatility.plugins.windows.lsadump module****volatility.plugins.windows.malfind module**

**class Malfind**(*context, config\_path, progress\_callback=None*)

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)



**volatility.plugins.windows.memmap module****class Memmap** (*context, config\_path, progress\_callback=None*)Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***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)**

## volatility.plugins.windows.modscan module

**class ModScan(context, config\_path, progress\_callback=None)**

Bases: `volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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** (`Optional[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)

**volatility.plugins.windows.modules module****class Modules** (*context, config\_path, progress\_callback=None*)

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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** (`Optional[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)
```

## volatility.plugins.windows.mutantscan module

**class MutantScan** (*context, config\_path, progress\_callback=None*)

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)

**volatility.plugins.windows.netscan module****class NetScan** (*context, config\_path, progress\_callback=None*)

Bases: *volatility.framework.interfaces.plugins.PluginInterface, volatility.plugins.timeliner.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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** *str*

**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)

## volatility.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*)

Bases: `object`

Class to maintain tag/size/index/type information about Pool header tags.

**class PoolHeaderScanner** (*module, constraint\_lookup, alignment*)

Bases: `volatility.framework.interfaces.layers.ScannerInterface`

**property context**

**Return type** `Optional[ContextInterface]`

**property layer\_name**

**Return type** `Optional[str]`

**thread\_safe** = False

**version** = (0, 0, 0)

**class PoolScanner** (*context, config\_path, progress\_callback=None*)

Bases: `volatility.framework.interfaces.plugins.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** (*Optional[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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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*)

Bases: *enum.IntEnum*

Class to maintain the different possible PoolTypes The values must be integer powers of 2.

**FREE** = 4

**NONPAGED** = 2

**PAGED** = 1

## volatility.plugins.windows.privileges module

**class Privs** (*\*args, \*\*kwargs*)

Bases: *volatility.framework.interfaces.plugins.PluginInterface*

Lists process token privileges

Args: 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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** `ContextInterface`

**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)

**volatility.plugins.windows.pslist module****class PsList** (*context, config\_path, progress\_callback=None*)Bases: *volatility.framework.interfaces.plugins.PluginInterface, volatility.plugins.timeliner.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***classmethod create\_name\_filter** (*name\_list=None*)

A factory for producing filter functions that filter based on a list of process names.

**Parameters** **name\_list** (*Optional[List[str]]*) – A list of process names that are acceptable, all other processes will be filtered out**Return type** *Callable[[ObjectInterface], bool]***Returns** Filter function for passing to the *list\_processes* method**classmethod create\_pid\_filter** (*pid\_list=None*)

A factory for producing filter functions that filter based on a list of process IDs.

**Parameters** **pid\_list** (*Optional[List[int]]*) – A list of process IDs that are acceptable, all other processes will be filtered out**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)**

## volatility.plugins.windows.psscan module

**class PsScan(context, config\_path, progress\_callback=None)**

Bases: *volatility.framework.interfaces.plugins.PluginInterface*, *volatility.plugins.timeliner.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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.can.<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** *Iterable[ObjectInterface]*

**Returns** A process object on virtual address layer

**volatility.plugins.windows.pstree module****class PsTree** (\*args, \*\*kwargs)Bases: *volatility.framework.interfaces.plugins.PluginInterface*

Plugin for listing processes in a tree based on their parent process ID.

Args: 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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***find\_level** (pid)

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)

## volatility.plugins.windows.ssdt module

**class SSDT** (*context, config\_path, progress\_callback=None*)

Bases: `volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)

### **volatility.plugins.windows.strings module**

**class** `Strings(context, config_path, progress_callback=None)`

Bases: `volatility.framework.interfaces.plugins.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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**generate\_mapping(layer\_name)**

Creates a reverse mapping between virtual addresses and physical addresses.

**Parameters** `layer_name` (`str`) – the layer to map against the string lines

**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** = (0, 0, 0)

## volatility.plugins.windows.svcscan module

## volatility.plugins.windows.symlinkscan module

**class** `SymlinkScan` (*context, config\_path, progress\_callback=None*)

Bases: `volatility.framework.interfaces.plugins.PluginInterface`, `volatility.plugins.timeliner.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)

**volatility.plugins.windows.vadinfo module****class VadInfo** (\*args, \*\*kwargs)Bases: *volatility.framework.interfaces.plugins.PluginInterface*

Lists process memory ranges.

Args: 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** = 0**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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***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=0*)

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)

## volatility.plugins.windows.vadparser module

## volatility.plugins.windows.verinfo module

**class VerInfo** (*context, config\_path, progress\_callback=None*)

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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** = (0, 0, 0)

**volatility.plugins.windows.virtmap module****class** **VirtMap** (*context, config\_path, progress\_callback=None*)Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***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

### volatility.plugins.banners module

**class Banners** (*context, config\_path, progress\_callback=None*)

Bases: `volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`

**property context**

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

**Return type** `ContextInterface`

**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)

## volatility.plugins.configwriter module

**class** `ConfigWriter(context, config_path, progress_callback=None)`

Bases: `volatility.framework.interfaces.plugins.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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**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)

**volatility.plugins.frameworkinfo module****class FrameworkInfo(context, config\_path, progress\_callback=None)**

Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict*

**property config\_path**

The configuration path on which this configurable lives.

**Return type** *str*

**property context**

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

**Return type** *ContextInterface*

**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)

## volatility.plugins.isfinfo module

**class** `IsfInfo(context, config_path, progress_callback=None)`

Bases: `volatility.framework.interfaces.plugins.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`

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`

**property** `config_path`

The configuration path on which this configurable lives.

**Return type** `str`

**property** `context`

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

**Return type** `ContextInterface`

**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** = (1, 0, 0)

**volatility.plugins.layerwriter module****class LayerWriter** (*context, config\_path, progress\_callback=None*)Bases: *volatility.framework.interfaces.plugins.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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** *HierarchicalDict***property config\_path**

The configuration path on which this configurable lives.

**Return type** *str***property context**

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

**Return type** *ContextInterface***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]`

**volatility.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)**Bases: `enum.IntEnum`

An enumeration.

**ACCESSED** = 3**CHANGED** = 4**CREATED** = 1**MODIFIED** = 2**class Timeliner(\*args, \*\*kwargs)**Bases: `volatility.framework.interfaces.plugins.PluginInterface`

Runs all relevant plugins that provide time related information and orders the results by time.

Args: 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**

The Hierarchical configuration Dictionary for this Configurable object.

**Return type** `HierarchicalDict`**property config\_path**

The configuration path on which this configurable lives.

**Return type** `str`**property context**

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

**Return type** `ContextInterface`**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)

## volatility.plugins.yarascan module

## volatility.schemas package

#### **create\_json\_hash(input, schema)**

Constructs the hash of the input and schema to create a unique identifier for a particular JSON file.

**Return type** *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`

### **volatility.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

volatility, 27  
volatility.cli, 28  
volatility.cli.text\_renderer, 41  
volatility.cli.volargparse, 43  
volatility.cli.volshell, 29  
volatility.cli.volshell.generic, 30  
volatility.cli.volshell.linux, 34  
volatility.cli.volshell.mac, 36  
volatility.cli.volshell.windows, 38  
volatility.framework, 44  
volatility.framework.automagic, 45  
volatility.framework.automagic.construct\_layers, 46  
volatility.framework.automagic.linux, 47  
volatility.framework.automagic.mac, 51  
volatility.framework.automagic.pdbscan, 54  
volatility.framework.automagic.stacker, 58  
volatility.framework.automagic.symbol\_cache, 60  
volatility.framework.automagic.symbol\_finder, 62  
volatility.framework.automagic.windows, 64  
volatility.framework.configuration, 70  
volatility.framework.configuration.requirements, 70  
volatility.framework.constants, 92  
volatility.framework.constants.linux, 93  
volatility.framework.constants.windows, 93  
volatility.framework.contexts, 93  
volatility.framework.exceptions, 406  
volatility.framework.interfaces, 99  
volatility.framework.interfaces.automagic, 99  
volatility.framework.interfaces.configuration, 101  
volatility.framework.interfaces.context, 112  
volatility.framework.interfaces.layers, 116  
volatility.framework.interfaces.objects, 122  
volatility.framework.interfaces.plugins, 126  
volatility.framework.interfaces.renderers, 129  
volatility.framework.interfaces.symbols, 132  
volatility.framework.layers, 140  
volatility.framework.layers.codecs, 140  
volatility.framework.layers.crash, 141  
volatility.framework.layers.elf, 147  
volatility.framework.layers.intel, 150  
volatility.framework.layers.lime, 166  
volatility.framework.layers.linear, 169  
volatility.framework.layers.msf, 171  
volatility.framework.layers.physical, 176  
volatility.framework.layers.qemu, 181  
volatility.framework.layers.registry, 184  
volatility.framework.layers.resources, 187  
volatility.framework.layers.scanners, 141  
volatility.framework.layers.scanners.multiregexp, 141  
volatility.framework.layers.segmented, 188  
volatility.framework.layers.vmware, 193  
volatility.framework.objects, 195  
volatility.framework.objects.templates, 234  
volatility.framework.objects.utility, 236  
volatility.framework.plugins, 236  
volatility.framework.renderers, 237  
volatility.framework.renderers.conversion,

240  
volatility.framework.renderers.format\_hints, 241  
241  
volatility.framework.symbols, 249  
volatility.framework.symbols.generic, 252  
volatility.framework.symbols.intermed, 377  
volatility.framework.symbols.linux, 253  
volatility.framework.symbols.linux.bash, 285  
volatility.framework.symbols.linux.extensions, 257  
volatility.framework.symbols.linux.extensions.bash, 279  
volatility.framework.symbols.linux.extensions.elf, 280  
volatility.framework.symbols.mac, 288  
volatility.framework.symbols.mac.extensions, 292  
volatility.framework.symbols.metadata, 404  
volatility.framework.symbols.native, 405  
volatility.framework.symbols.windows, 312  
volatility.framework.symbols.windows.extensions, 315  
volatility.framework.symbols.windows.extensions.kdbg, 346  
volatility.framework.symbols.windows.extensions.network, 348  
volatility.framework.symbols.windows.extensions.pcap, 348  
volatility.framework.symbols.windows.extensions.pyop, 352  
volatility.framework.symbols.windows.extensions.registry, 361  
volatility.framework.symbols.windows.extensions.serve\_plugins, 370  
volatility.framework.symbols.windows.pdbutil, 373  
volatility.framework.symbols.windows.pdbutil, 375  
volatility.framework.symbols.windows.versions, 377  
volatility.framework.symbols.wrappers, 406  
volatility.plugins, 408  
volatility.plugins.banners, 533  
volatility.plugins.configwriter, 535  
volatility.plugins.frameworkinfo, 536  
volatility.plugins.isinfo, 538  
volatility.plugins.layerwriter, 540  
volatility.plugins.linux, 408  
volatility.plugins.linux.bash, 409  
volatility.plugins.linux.check\_afinfo, 410  
volatility.plugins.linux.check\_creds, 412  
volatility.plugins.linux.check\_idt, 413  
volatility.plugins.linux.check\_modules, 415  
volatility.plugins.linux.check\_syscall, 416  
volatility.plugins.linux.elfs, 418  
volatility.plugins.linux.keyboard\_notifiers, 420  
volatility.plugins.linux.lsmmod, 421  
volatility.plugins.linux.lsof, 423  
volatility.plugins.linux.malfind, 424  
volatility.plugins.linux.proc, 426  
volatility.plugins.linux.pslist, 427  
volatility.plugins.linux.pstree, 429  
volatility.plugins.linux.tty\_check, 431  
volatility.plugins.mac, 433  
volatility.plugins.mac.bash, 433  
volatility.plugins.mac.check\_syscall, 435  
volatility.plugins.mac.check\_sysctl, 436  
volatility.plugins.mac.check\_trap\_table, 438  
volatility.plugins.mac.ifconfig, 439  
volatility.plugins.mac.kauth\_listeners, 441  
volatility.plugins.mac.kauth\_scopes, 442  
volatility.plugins.mac.kevents, 444  
volatility.plugins.mac.list\_files, 446  
volatility.plugins.mac.lsmmod, 447  
volatility.plugins.mac.lsof, 449  
volatility.plugins.mac.malfind, 450  
volatility.plugins.mac.mount, 452  
volatility.plugins.mac.netstat, 454  
volatility.plugins.mac.proc\_maps, 455  
volatility.plugins.mac.psaux, 457  
volatility.plugins.mac.pslist, 459  
volatility.plugins.mac.pstree, 462  
volatility.plugins.mac.socket\_filters, 463  
volatility.plugins.mac.timers, 465  
volatility.plugins.mac.trustedbsd, 466  
volatility.plugins.mac.vfsevents, 468  
volatility.plugins.timeliner, 542  
volatility.plugins.windows, 469  
volatility.plugins.windows.bigpools, 477  
volatility.plugins.windows.cmdline, 479  
volatility.plugins.windows.dlllist, 481  
volatility.plugins.windows.driverirp, 483



`volatility.plugins.windows.driverscan`,  
484  
`volatility.plugins.windows.dumpfiles`,  
486  
`volatility.plugins.windows.envvars`, 488  
`volatility.plugins.windows.filescan`, 490  
`volatility.plugins.windows.getservicesids`,  
491  
`volatility.plugins.windows.getsids`, 493  
`volatility.plugins.windows.handles`, 494  
`volatility.plugins.windows.info`, 496  
`volatility.plugins.windows.malfind`, 499  
`volatility.plugins.windows.memmap`, 501  
`volatility.plugins.windows.modscan`, 502  
`volatility.plugins.windows.modules`, 504  
`volatility.plugins.windows.mutantscan`,  
507  
`volatility.plugins.windows.netscan`, 508  
`volatility.plugins.windows.poolscanner`,  
511  
`volatility.plugins.windows.privileges`,  
514  
`volatility.plugins.windows.pslist`, 516  
`volatility.plugins.windows.psscan`, 518  
`volatility.plugins.windows.pstree`, 521  
`volatility.plugins.windows.registry`, 469  
`volatility.plugins.windows.registry.hivelist`,  
470  
`volatility.plugins.windows.registry.hivescan`,  
472  
`volatility.plugins.windows.registry.printkey`,  
474  
`volatility.plugins.windows.registry.userassist`,  
476  
`volatility.plugins.windows.ssdt`, 522  
`volatility.plugins.windows.strings`, 524  
`volatility.plugins.windows.symlinkscan`,  
526  
`volatility.plugins.windows.vadinfo`, 528  
`volatility.plugins.windows.verinfo`, 530  
`volatility.plugins.windows.virtmap`, 532  
`volatility.schemas`, 543  
`volatility.symbols`, 544



## A

- `access_string()` (*FILE\_OBJECT* method), 326
- `ACCESSED` (*TimeLinerType* attribute), 542
- `add_argument()` (*HelpfulArgParser* method), 43
- `add_argument_group()` (*HelpfulArgParser* method), 43
- `add_layer()` (*Context* method), 93
- `add_layer()` (*ContextInterface* method), 112
- `add_layer()` (*LayerContainer* method), 118
- `add_mutually_exclusive_group()` (*HelpfulArgParser* method), 43
- `add_parent()` (*JarHandler* method), 187
- `add_parser()` (*HelpfulSubparserAction* method), 44
- `add_pattern()` (*MultiRegexp* method), 141
- `add_process_layer()` (*EPROCESS* method), 321
- `add_process_layer()` (*proc* method), 298
- `add_process_layer()` (*task\_struct* method), 274
- `add_requirement()` (*BooleanRequirement* method), 70
- `add_requirement()` (*BytesRequirement* method), 71
- `add_requirement()` (*ChoiceRequirement* method), 73
- `add_requirement()` (*ClassRequirement* method), 101
- `add_requirement()` (*ComplexListRequirement* method), 74
- `add_requirement()` (*ConfigurableRequirementInterface* method), 104
- `add_requirement()` (*ConstructableRequirementInterface* method), 106
- `add_requirement()` (*IntRequirement* method), 76
- `add_requirement()` (*LayerListRequirement* method), 77
- `add_requirement()` (*ListRequirement* method), 79
- `add_requirement()` (*MultiRequirement* method), 81
- `add_requirement()` (*PluginRequirement* method), 82
- `add_requirement()` (*RequirementInterface* method), 109
- `add_requirement()` (*SimpleTypeRequirement* method), 111
- `add_requirement()` (*StringRequirement* method), 84
- `add_requirement()` (*SymbolTableRequirement* method), 85
- `add_requirement()` (*TranslationLayerRequirement* method), 87
- `add_requirement()` (*URIRequirement* method), 89
- `add_requirement()` (*VersionRequirement* method), 90
- `add_subparsers()` (*HelpfulArgParser* method), 43
- `address()` (*SymbolInterface* property), 136
- `address_mask()` (*BufferDataLayer* property), 176
- `address_mask()` (*DataLayerInterface* property), 116
- `address_mask()` (*Elf64Layer* property), 147
- `address_mask()` (*FileLayer* property), 178
- `address_mask()` (*Intel* property), 150
- `address_mask()` (*Intel32e* property), 152
- `address_mask()` (*IntelPAE* property), 154
- `address_mask()` (*LimeLayer* property), 166
- `address_mask()` (*LinearlyMappedLayer* property), 169
- `address_mask()` (*NonLinearlySegmentedLayer* property), 188
- `address_mask()` (*PdbMSFStream* property), 171
- `address_mask()` (*PdbMultiStreamFormat* property), 174
- `address_mask()` (*QemuSuspendLayer* property), 182
- `address_mask()` (*RegistryHive* property), 184
- `address_mask()` (*SegmentedLayer* property), 190
- `address_mask()` (*TranslationLayerInterface* property), 120
- `address_mask()` (*VmwareLayer* property), 193
- `address_mask()` (*WindowsCrashDump32Layer* property), 141
- `address_mask()` (*WindowsCrashDump64Layer* property), 144
- `address_mask()` (*WindowsIntel* property), 156
- `address_mask()` (*WindowsIntel32e* property), 159
- `address_mask()` (*WindowsIntelPAE* property), 161
- `address_mask()` (*WindowsMixin* property), 163
- `AggregateType` (class in *volatility.framework.objects*), 195
- `AggregateType.VolTemplateProxy` (class in

*volatility.framework.objects*), 195  
all\_filters (*Kevents* attribute), 444  
append() (*SymbolSpace* method), 250  
append() (*SymbolSpaceInterface* method), 136  
args (*ElfFormatException* attribute), 149  
args (*InvalidAddressException* attribute), 406  
args (*LayerException* attribute), 407  
args (*LimeFormatException* attribute), 166  
args (*MissingModuleException* attribute), 407  
args (*PagedInvalidAddressException* attribute), 407  
args (*PDBFormatException* attribute), 171  
args (*PluginRequirementException* attribute), 407  
args (*PluginVersionException* attribute), 407  
args (*RegistryFormatException* attribute), 184  
args (*RegistryInvalidIndex* attribute), 187  
args (*SwappedInvalidAddressException* attribute), 407  
args (*SymbolError* attribute), 408  
args (*SymbolSpaceError* attribute), 408  
args (*UnsatisfiedException* attribute), 408  
args (*VmwareFormatException* attribute), 193  
args (*VolatilityException* attribute), 408  
args (*WindowsCrashDumpFormatException* attribute), 146  
Array (class in *volatility.framework.objects*), 197  
Array.VolTemplateProxy (class in *volatility.framework.objects*), 197  
array\_of\_pointers() (in module *volatility.framework.objects.utility*), 236  
array\_to\_string() (in module *volatility.framework.objects.utility*), 236  
as\_integer\_ratio() (*Float* method), 215  
ascending (*ColumnSortKey* attribute), 130, 237  
AUTOMAGIC\_CONFIG\_PATH (in module *volatility.framework.constants*), 92  
AutomagicInterface (class in *volatility.framework.interfaces.automagic*), 99  
available() (in module *volatility.framework.automagic*), 45

## B

BANG (in module *volatility.framework.constants*), 92  
banner\_cache (*LinuxSymbolFinder* attribute), 49  
banner\_cache (*MacSymbolFinder* attribute), 53  
banner\_cache (*SymbolFinder* attribute), 62  
banner\_config\_key (*LinuxSymbolFinder* attribute), 49  
banner\_config\_key (*MacSymbolFinder* attribute), 53  
banner\_config\_key (*SymbolFinder* attribute), 62  
banner\_path (*LinuxBannerCache* attribute), 47  
banner\_path (*MacBannerCache* attribute), 51  
banner\_path (*SymbolBannerCache* attribute), 60  
Banners (class in *volatility.plugins.banners*), 533  
banners() (*LinuxSymbolFinder* property), 49

banners() (*MacSymbolFinder* property), 53  
banners() (*SymbolFinder* property), 62  
base\_types (*TreeGrid* attribute), 130, 238  
BaseAbsentValue (class in *volatility.framework.interfaces.renderers*), 129  
BaseSymbolTableInterface (class in *volatility.framework.interfaces.symbols*), 132  
Bash (class in *volatility.plugins.linux.bash*), 409  
Bash (class in *volatility.plugins.mac.bash*), 433  
BashIntermedSymbols (class in *volatility.framework.symbols.linux.bash*), 285  
BigPools (class in *volatility.plugins.windows.bigpools*), 477  
Bin (class in *volatility.framework.renderers.format\_hints*), 241  
bit\_length() (*Bin* method), 241  
bit\_length() (*BitField* method), 199  
bit\_length() (*Boolean* method), 201  
bit\_length() (*Char* method), 208  
bit\_length() (*Enumeration* method), 212  
bit\_length() (*Hex* method), 242  
bit\_length() (*Integer* method), 218  
bit\_length() (*Pointer* method), 221  
BitField (class in *volatility.framework.objects*), 198  
BitField.VolTemplateProxy (class in *volatility.framework.objects*), 199  
bits\_per\_register (*Intel* attribute), 150  
bits\_per\_register (*Intel32e* attribute), 152  
bits\_per\_register (*IntelPAE* attribute), 154  
bits\_per\_register (*WindowsIntel* attribute), 156  
bits\_per\_register (*WindowsIntel32e* attribute), 159  
bits\_per\_register (*WindowsIntelPAE* attribute), 161  
bits\_per\_register (*WindowsMixin* attribute), 163  
Boolean (class in *volatility.framework.objects*), 201  
Boolean.VolTemplateProxy (class in *volatility.framework.objects*), 201  
BooleanRequirement (class in *volatility.framework.configuration.requirements*), 70  
branch() (*HierarchicalDict* method), 108  
BufferDataLayer (class in *volatility.framework.layers.physical*), 176  
build\_configuration() (*AutomagicInterface* method), 99  
build\_configuration() (*Banners* method), 533  
build\_configuration() (*Bash* method), 409, 433  
build\_configuration() (*BashIntermedSymbols* method), 285  
build\_configuration() (*BigPools* method), 477  
build\_configuration() (*BufferDataLayer* method), 176  
build\_configuration() (*Check\_afinfo* method),

410  
 build\_configuration() (*Check\_creds method*), 412  
 build\_configuration() (*Check\_idt method*), 413  
 build\_configuration() (*Check\_modules method*), 415  
 build\_configuration() (*Check\_syscall method*), 417, 435  
 build\_configuration() (*Check\_sysctl method*), 436  
 build\_configuration() (*Check\_trap\_table method*), 438  
 build\_configuration() (*CmdLine method*), 479  
 build\_configuration() (*ComplexListRequirement method*), 74  
 build\_configuration() (*ConfigurableInterface method*), 103  
 build\_configuration() (*ConfigurableRequirementInterface method*), 104  
 build\_configuration() (*ConfigWriter method*), 535  
 build\_configuration() (*ConstructionMagic method*), 46  
 build\_configuration() (*DataLayerInterface method*), 116  
 build\_configuration() (*DllList method*), 481  
 build\_configuration() (*DriverIrp method*), 483  
 build\_configuration() (*DriverScan method*), 484  
 build\_configuration() (*DumpFiles method*), 486  
 build\_configuration() (*Elf64Layer method*), 147  
 build\_configuration() (*Elfs method*), 418  
 build\_configuration() (*Envars method*), 488  
 build\_configuration() (*FileLayer method*), 179  
 build\_configuration() (*FileScan method*), 490  
 build\_configuration() (*FrameworkInfo method*), 536  
 build\_configuration() (*GetServiceSIDs method*), 491  
 build\_configuration() (*GetSIDs method*), 493  
 build\_configuration() (*Handles method*), 494  
 build\_configuration() (*HiveList method*), 470  
 build\_configuration() (*HiveScan method*), 472  
 build\_configuration() (*Ifconfig method*), 439  
 build\_configuration() (*Info method*), 497  
 build\_configuration() (*Intel method*), 150  
 build\_configuration() (*Intel32e method*), 152  
 build\_configuration() (*IntelPAE method*), 154  
 build\_configuration() (*IntermediateSymbolTable method*), 381  
 build\_configuration() (*IsfInfo method*), 538  
 build\_configuration() (*ISFormatTable method*), 378  
 build\_configuration() (*Kauth\_listeners method*), 441  
 build\_configuration() (*Kauth\_scopes method*), 442  
 build\_configuration() (*KernelPDBScanner method*), 55  
 build\_configuration() (*Kevents method*), 444  
 build\_configuration() (*Keyboard\_notifiers method*), 420  
 build\_configuration() (*LayerListRequirement method*), 77  
 build\_configuration() (*LayerStacker method*), 58  
 build\_configuration() (*LayerWriter method*), 540  
 build\_configuration() (*LimeLayer method*), 166  
 build\_configuration() (*LinearlyMappedLayer method*), 169  
 build\_configuration() (*LinuxBannerCache method*), 47  
 build\_configuration() (*LinuxKernelIntermedSymbols method*), 254  
 build\_configuration() (*LinuxSymbolFinder method*), 49  
 build\_configuration() (*List\_Files method*), 446  
 build\_configuration() (*Lsmode method*), 421, 447  
 build\_configuration() (*Lsof method*), 423, 449  
 build\_configuration() (*MacBannerCache method*), 51  
 build\_configuration() (*MacKernelIntermedSymbols method*), 288  
 build\_configuration() (*MacSymbolFinder method*), 53  
 build\_configuration() (*Malfind method*), 425, 450, 499  
 build\_configuration() (*Maps method*), 426, 455  
 build\_configuration() (*Memmap method*), 501  
 build\_configuration() (*ModScan method*), 502  
 build\_configuration() (*Modules method*), 505  
 build\_configuration() (*Mount method*), 452  
 build\_configuration() (*MutantScan method*), 507  
 build\_configuration() (*NetScan method*), 509  
 build\_configuration() (*Netstat method*), 454  
 build\_configuration() (*NonLinearlySegmentedLayer method*), 188  
 build\_configuration() (*PdbMSFStream method*), 171  
 build\_configuration() (*PdbMultiStreamFormat method*), 174  
 build\_configuration() (*PluginInterface method*), 128  
 build\_configuration() (*PoolScanner method*),

- 511  
build\_configuration() (*PrintKey method*), 474  
build\_configuration() (*Privs method*), 514  
build\_configuration() (*Psaux method*), 457  
build\_configuration() (*PsList method*), 428, 459, 516  
build\_configuration() (*PsScan method*), 518  
build\_configuration() (*PsTree method*), 429, 462, 521  
build\_configuration() (*QemuSuspendLayer method*), 182  
build\_configuration() (*RegistryHive method*), 184  
build\_configuration() (*SegmentedLayer method*), 190  
build\_configuration() (*Socket\_filters method*), 463  
build\_configuration() (*SSDT method*), 522  
build\_configuration() (*Strings method*), 524  
build\_configuration() (*SymbolBannerCache method*), 60  
build\_configuration() (*SymbolFinder method*), 62  
build\_configuration() (*SymbolTableInterface method*), 137  
build\_configuration() (*SymbolTableRequirement method*), 85  
build\_configuration() (*SymlinkScan method*), 526  
build\_configuration() (*Timeliner method*), 542  
build\_configuration() (*Timers method*), 465  
build\_configuration() (*TranslationLayerInterface method*), 120  
build\_configuration() (*TranslationLayerRequirement method*), 87  
build\_configuration() (*Trustedbsd method*), 466  
build\_configuration() (*tty\_check method*), 431  
build\_configuration() (*UserAssist method*), 476  
build\_configuration() (*VadInfo method*), 528  
build\_configuration() (*VerInfo method*), 530  
build\_configuration() (*Version1Format method*), 384  
build\_configuration() (*Version2Format method*), 386  
build\_configuration() (*Version3Format method*), 389  
build\_configuration() (*Version4Format method*), 391  
build\_configuration() (*Version5Format method*), 394  
build\_configuration() (*Version6Format method*), 397  
build\_configuration() (*Version7Format method*), 399  
build\_configuration() (*Version8Format method*), 402  
build\_configuration() (*VFSevents method*), 468  
build\_configuration() (*VirtMap method*), 532  
build\_configuration() (*VmwareLayer method*), 193  
build\_configuration() (*Volshell method*), 32, 34, 36, 38  
build\_configuration() (*WindowsCrash-Dump32Layer method*), 141  
build\_configuration() (*WindowsCrash-Dump64Layer method*), 144  
build\_configuration() (*WindowsIntel method*), 157  
build\_configuration() (*WindowsIntel32e method*), 159  
build\_configuration() (*WindowsIntelPAE method*), 161  
build\_configuration() (*WindowsKernelInter-medSymbols method*), 312  
build\_configuration() (*WindowsMixin method*), 163  
build\_configuration() (*WinSwapLayers method*), 66  
build\_configuration() (*WintelHelper method*), 68  
build\_module\_collection() (*SSDT class method*), 522  
builtin\_constraints() (*PoolScanner static method*), 512  
byteorder() (*DataFormatInfo property*), 211  
Bytes (*class in volatility.framework.objects*), 203  
Bytes.VolTemplateProxy (*class in volatility.framework.objects*), 203  
BytesRequirement (*class in volatility.framework.configuration.requirements*), 71  
BytesScanner (*class in volatility.framework.layers.scanners*), 140
- ## C
- CACHE\_PATH (*in module volatility.framework.constants*), 92  
cached\_strtab() (*elf\_sym property*), 284  
capitalize() (*Bytes method*), 203  
capitalize() (*HexBytes method*), 242  
capitalize() (*MultiTypeData method*), 246  
capitalize() (*String method*), 225  
cascadeCloseFile() (*in module volatility.framework.layers.resources*), 188  
casefold() (*String method*), 225  
cast() (*AggregateType method*), 196  
cast() (*Array method*), 197  
cast() (*BitField method*), 199



- `cast ()` (*Boolean method*), 201
- `cast ()` (*Bytes method*), 204
- `cast ()` (*Char method*), 208
- `cast ()` (*ClassType method*), 210
- `cast ()` (*CM\_KEY\_BODY method*), 363
- `cast ()` (*CM\_KEY\_NODE method*), 364
- `cast ()` (*CM\_KEY\_VALUE method*), 366
- `cast ()` (*CMHIVE method*), 361
- `cast ()` (*CONTROL\_AREA method*), 316
- `cast ()` (*dentry method*), 257
- `cast ()` (*DEVICE\_OBJECT method*), 318
- `cast ()` (*DRIVER\_OBJECT method*), 319
- `cast ()` (*elf method*), 281
- `cast ()` (*elf\_phdr method*), 282
- `cast ()` (*elf\_sym method*), 284
- `cast ()` (*Enumeration method*), 212
- `cast ()` (*EPROCESS method*), 321
- `cast ()` (*ETHREAD method*), 323
- `cast ()` (*EX\_FAST\_REF method*), 325
- `cast ()` (*ExecutiveObject method*), 352
- `cast ()` (*FILE\_OBJECT method*), 326
- `cast ()` (*fileglob method*), 292
- `cast ()` (*files\_struct method*), 259
- `cast ()` (*Float method*), 215
- `cast ()` (*fs\_struct method*), 260
- `cast ()` (*Function method*), 217
- `cast ()` (*GenericIntelProcess method*), 252
- `cast ()` (*hist\_entry method*), 279
- `cast ()` (*HMAP\_ENTRY method*), 368
- `cast ()` (*ifnet method*), 294
- `cast ()` (*IMAGE\_DOS\_HEADER method*), 348
- `cast ()` (*IMAGE\_NT\_HEADERS method*), 350
- `cast ()` (*inpcb method*), 295
- `cast ()` (*Integer method*), 219
- `cast ()` (*kauth\_scope method*), 297
- `cast ()` (*KDDEBUGGER\_DATA64 method*), 347
- `cast ()` (*KMUTANT method*), 328
- `cast ()` (*kobject method*), 262
- `cast ()` (*KSYSTEM\_TIME method*), 329
- `cast ()` (*KTHREAD method*), 331
- `cast ()` (*LIST\_ENTRY method*), 332
- `cast ()` (*list\_head method*), 263
- `cast ()` (*mm\_struct method*), 265
- `cast ()` (*MMVAD method*), 334
- `cast ()` (*MMVAD\_SHORT method*), 336
- `cast ()` (*module method*), 266
- `cast ()` (*mount method*), 268
- `cast ()` (*OBJECT\_HEADER method*), 354
- `cast ()` (*OBJECT\_SYMBOLIC\_LINK method*), 338
- `cast ()` (*ObjectInterface method*), 124
- `cast ()` (*Pointer method*), 221
- `cast ()` (*POOL\_HEADER method*), 356
- `cast ()` (*POOL\_HEADER\_VISTA method*), 357
- `cast ()` (*POOL\_TRACKER\_BIG\_PAGES method*), 359
- `cast ()` (*PrimitiveObject method*), 223
- `cast ()` (*proc method*), 298
- `cast ()` (*qstr method*), 270
- `cast ()` (*queue\_entry method*), 300
- `cast ()` (*SERVICE\_HEADER method*), 370
- `cast ()` (*SERVICE\_RECORD method*), 372
- `cast ()` (*SHARED\_CACHE\_MAP method*), 340
- `cast ()` (*sockaddr method*), 302
- `cast ()` (*sockaddr\_dl method*), 303
- `cast ()` (*socket method*), 305
- `cast ()` (*String method*), 225
- `cast ()` (*struct\_file method*), 271
- `cast ()` (*StructType method*), 230
- `cast ()` (*super\_block method*), 273
- `cast ()` (*sysctl\_oid method*), 306
- `cast ()` (*task\_struct method*), 274
- `cast ()` (*TOKEN method*), 342
- `cast ()` (*UNICODE\_STRING method*), 343
- `cast ()` (*UnionType method*), 232
- `cast ()` (*VACB method*), 345
- `cast ()` (*vfsmount method*), 276
- `cast ()` (*vm\_area\_struct method*), 277
- `cast ()` (*vm\_map\_entry method*), 308
- `cast ()` (*vm\_map\_object method*), 310
- `cast ()` (*vnode method*), 311
- `cast ()` (*Void method*), 233
- `center ()` (*Bytes method*), 204
- `center ()` (*HexBytes method*), 243
- `center ()` (*MultiTypeData method*), 246
- `center ()` (*String method*), 225
- `change_layer ()` (*Volshell method*), 32, 34, 36, 38
- `change_process ()` (*Volshell method*), 39
- `change_task ()` (*Volshell method*), 34, 36
- `CHANGED` (*TimeLinerType attribute*), 542
- `Char` (*class in volatility.framework.objects*), 207
- `Char.VolTemplateProxy` (*class in volatility.framework.objects*), 208
- `Check_afinfo` (*class in volatility.plugins.linux.check\_afinfo*), 410
- `Check_creds` (*class in volatility.plugins.linux.check\_creds*), 412
- `check_cycles ()` (*LayerContainer method*), 118
- `check_header ()` (*WindowsCrashDump32Layer class method*), 141
- `check_header ()` (*WindowsCrashDump64Layer class method*), 144
- `Check_idt` (*class in volatility.plugins.linux.check\_idt*), 413
- `check_kernel_offset ()` (*KernelPDBScanner method*), 55
- `Check_modules` (*class in volatility.plugins.linux.check\_modules*), 415
- `Check_syscall` (*class in volatility.plugins.linux.check\_syscall*), 416

`Check_syscall` (class in `volatility.plugins.mac.check_syscall`), 435

`Check_sysctl` (class in `volatility.plugins.mac.check_sysctl`), 436

`Check_trap_table` (class in `volatility.plugins.mac.check_trap_table`), 438

`children()` (`AggregateType.VolTemplateProxy` class method), 196

`children()` (`Array.VolTemplateProxy` class method), 197

`children()` (`BitField.VolTemplateProxy` class method), 199

`children()` (`Boolean.VolTemplateProxy` class method), 201

`children()` (`Bytes.VolTemplateProxy` class method), 203

`children()` (`Char.VolTemplateProxy` class method), 208

`children()` (`ClassType.VolTemplateProxy` class method), 210

`children()` (`CM_KEY_BODY.VolTemplateProxy` class method), 363

`children()` (`CM_KEY_NODE.VolTemplateProxy` class method), 364

`children()` (`CM_KEY_VALUE.VolTemplateProxy` class method), 366

`children()` (`CMHIVE.VolTemplateProxy` class method), 361

`children()` (`CONTROL_AREA.VolTemplateProxy` class method), 315

`children()` (`dentry.VolTemplateProxy` class method), 257

`children()` (`DEVICE_OBJECT.VolTemplateProxy` class method), 317

`children()` (`DRIVER_OBJECT.VolTemplateProxy` class method), 319

`children()` (`elf.VolTemplateProxy` class method), 280

`children()` (`elf_phdr.VolTemplateProxy` class method), 282

`children()` (`elf_sym.VolTemplateProxy` class method), 284

`children()` (`Enumeration.VolTemplateProxy` class method), 212

`children()` (`EPROCESS.VolTemplateProxy` class method), 320

`children()` (`ETHREAD.VolTemplateProxy` class method), 323

`children()` (`EX_FAST_REF.VolTemplateProxy` class method), 324

`children()` (`ExecutiveObject.VolTemplateProxy` class method), 352

`children()` (`FILE_OBJECT.VolTemplateProxy` class method), 326

`children()` (`fileglob.VolTemplateProxy` class method), 292

`children()` (`files_struct.VolTemplateProxy` class method), 258

`children()` (`Float.VolTemplateProxy` class method), 214

`children()` (`fs_struct.VolTemplateProxy` class method), 260

`children()` (`Function.VolTemplateProxy` class method), 217

`children()` (`GenericIntelProcess.VolTemplateProxy` class method), 252

`children()` (`hist_entry.VolTemplateProxy` class method), 279

`children()` (`HMAP_ENTRY.VolTemplateProxy` class method), 367

`children()` (`ifnet.VolTemplateProxy` class method), 293

`children()` (`IMAGE_DOS_HEADER.VolTemplateProxy` class method), 348

`children()` (`IMAGE_NT_HEADERS.VolTemplateProxy` class method), 350

`children()` (`inpcb.VolTemplateProxy` class method), 295

`children()` (`Integer.VolTemplateProxy` class method), 218

`children()` (`kauth_scope.VolTemplateProxy` class method), 296

`children()` (`KDDEBUGGER_DATA64.VolTemplateProxy` class method), 346

`children()` (`KMUTANT.VolTemplateProxy` class method), 327

`children()` (`kobject.VolTemplateProxy` class method), 261

`children()` (`KSYSTEM_TIME.VolTemplateProxy` class method), 329

`children()` (`KTHREAD.VolTemplateProxy` class method), 331

`children()` (`LIST_ENTRY.VolTemplateProxy` class method), 332

`children()` (`list_head.VolTemplateProxy` class method), 263

`children()` (`mm_struct.VolTemplateProxy` class method), 264

`children()` (`MMVAD.VolTemplateProxy` class method), 334

`children()` (`MMVAD_SHORT.VolTemplateProxy` class method), 336

`children()` (`module.VolTemplateProxy` class method), 266

`children()` (`mount.VolTemplateProxy` class method), 268

`children()` (`OBJECT_HEADER.VolTemplateProxy` class method), 354



[children\(\) \(OBJECT\\_SYMBOLIC\\_LINK.VolTemplateProxy class method\), 338](#)  
[children\(\) \(ObjectInterface.VolTemplateProxy class method\), 123](#)  
[children\(\) \(ObjectTemplate property\), 234](#)  
[children\(\) \(Pointer.VolTemplateProxy class method\), 220](#)  
[children\(\) \(POOL\\_HEADER.VolTemplateProxy class method\), 355](#)  
[children\(\) \(POOL\\_HEADER\\_VISTA.VolTemplateProxy class method\), 357](#)  
[children\(\) \(POOL\\_TRACKER\\_BIG\\_PAGES.VolTemplateProxy class method\), 359](#)  
[children\(\) \(PrimitiveObject.VolTemplateProxy class method\), 223](#)  
[children\(\) \(proc.VolTemplateProxy class method\), 298](#)  
[children\(\) \(qstr.VolTemplateProxy class method\), 269](#)  
[children\(\) \(queue\\_entry.VolTemplateProxy class method\), 300](#)  
[children\(\) \(ReferenceTemplate property\), 235](#)  
[children\(\) \(SERVICE\\_HEADER.VolTemplateProxy class method\), 370](#)  
[children\(\) \(SERVICE\\_RECORD.VolTemplateProxy class method\), 371](#)  
[children\(\) \(SHARED\\_CACHE\\_MAP.VolTemplateProxy class method\), 340](#)  
[children\(\) \(sockaddr.VolTemplateProxy class method\), 301](#)  
[children\(\) \(sockaddr\\_dl.VolTemplateProxy class method\), 303](#)  
[children\(\) \(socket.VolTemplateProxy class method\), 304](#)  
[children\(\) \(String.VolTemplateProxy class method\), 224](#)  
[children\(\) \(struct\\_file.VolTemplateProxy class method\), 271](#)  
[children\(\) \(StructType.VolTemplateProxy class method\), 230](#)  
[children\(\) \(super\\_block.VolTemplateProxy class method\), 272](#)  
[children\(\) \(SymbolSpace.UnresolvedTemplate property\), 249](#)  
[children\(\) \(sysctl\\_oid.VolTemplateProxy class method\), 306](#)  
[children\(\) \(task\\_struct.VolTemplateProxy class method\), 274](#)  
[children\(\) \(Template property\), 125](#)  
[children\(\) \(TOKEN.VolTemplateProxy class method\), 341](#)  
[children\(\) \(TreeGrid method\), 130, 238](#)  
[children\(\) \(UNICODE\\_STRING.VolTemplateProxy class method\), 343](#)  
[children\(\) \(UnionType.VolTemplateProxy class method\), 231](#)  
[children\(\) \(VACB.VolTemplateProxy class method\), 345](#)  
[children\(\) \(vfsmount.VolTemplateProxy class method\), 275](#)  
[children\(\) \(vm\\_area\\_struct.VolTemplateProxy class method\), 277](#)  
[children\(\) \(vm\\_map\\_entry.VolTemplateProxy class method\), 308](#)  
[children\(\) \(vm\\_map\\_object.VolTemplateProxy class method\), 309](#)  
[children\(\) \(vnode.VolTemplateProxy class method\), 311](#)  
[children\(\) \(Void.VolTemplateProxy class method\), 233](#)  
[ChoiceRequirement \(class in volatility.framework.configuration.requirements\), 72](#)  
[choices\(\) \(Enumeration property\), 213](#)  
[choices\(\) \(Flags property\), 406](#)  
[choose\\_automagic\(\) \(in module volatility.framework.automagic\), 45](#)  
[choose\\_os\\_stackables\(\) \(in module volatility.framework.automagic.stackers\), 60](#)  
[class\\_subclasses\(\) \(in module volatility.framework\), 44](#)  
[classproperty \(class in volatility\), 27](#)  
[ClassRequirement \(class in volatility.framework.interfaces.configuration\), 101](#)  
[ClassType \(class in volatility.framework.objects\), 210](#)  
[ClassType.VolTemplateProxy \(class in volatility.framework.objects\), 210](#)  
[clear\\_cache\(\) \(in module volatility.framework\), 44](#)  
[clear\\_symbol\\_cache\(\) \(BaseSymbolTableInterface method\), 133](#)  
[clear\\_symbol\\_cache\(\) \(BashIntermedSymbols method\), 286](#)  
[clear\\_symbol\\_cache\(\) \(IntermediateSymbolTable method\), 381](#)  
[clear\\_symbol\\_cache\(\) \(ISFormatTable method\), 378](#)  
[clear\\_symbol\\_cache\(\) \(LinuxKernelIntermedSymbols method\), 254](#)  
[clear\\_symbol\\_cache\(\) \(MacKernelIntermedSymbols method\), 289](#)  
[clear\\_symbol\\_cache\(\) \(NativeTable method\), 405](#)  
[clear\\_symbol\\_cache\(\) \(NativeTableInterface method\), 134](#)  
[clear\\_symbol\\_cache\(\) \(SymbolSpace method\), 250](#)  
[clear\\_symbol\\_cache\(\) \(SymbolSpaceInterface method\), 136](#)  
[clear\\_symbol\\_cache\(\) \(SymbolTableInterface](#)

method), 138

clear\_symbol\_cache() (*Version1Format method*), 384

clear\_symbol\_cache() (*Version2Format method*), 386

clear\_symbol\_cache() (*Version3Format method*), 389

clear\_symbol\_cache() (*Version4Format method*), 391

clear\_symbol\_cache() (*Version5Format method*), 394

clear\_symbol\_cache() (*Version6Format method*), 397

clear\_symbol\_cache() (*Version7Format method*), 399

clear\_symbol\_cache() (*Version8Format method*), 402

clear\_symbol\_cache() (*WindowsKernelIntermedSymbols method*), 313

CLI\_NAME (*CommandLine attribute*), 28

CLI\_NAME (*VolShell attribute*), 29

CLIRenderer (*class in volatility.cli.text\_renderer*), 41

clone() (*Context method*), 94

clone() (*ContextInterface method*), 113

clone() (*HierarchicalDict method*), 108

clone() (*ObjectTemplate method*), 234

clone() (*ReferenceTemplate method*), 235

clone() (*SymbolSpace.UnresolvedTemplate method*), 250

clone() (*Template method*), 125

close() (*FileHandlerInterface method*), 126

close() (*JarHandler method*), 187

close() (*NullFileHandler method*), 30

closed (*FileHandlerInterface attribute*), 126

closed (*NullFileHandler attribute*), 30

cls() (*ClassRequirement property*), 102

CM\_KEY\_BODY (*class in volatility.framework.symbols.windows.extensions.registry*), 362

CM\_KEY\_BODY.VolTemplateProxy (*class in volatility.framework.symbols.windows.extensions.registry*), 362

CM\_KEY\_NODE (*class in volatility.framework.symbols.windows.extensions.registry*), 364

CM\_KEY\_NODE.VolTemplateProxy (*class in volatility.framework.symbols.windows.extensions.registry*), 364

CM\_KEY\_VALUE (*class in volatility.framework.symbols.windows.extensions.registry*), 366

CM\_KEY\_VALUE.VolTemplateProxy (*class in volatility.framework.symbols.windows.extensions.registry*), 366

CmdLine (*class in volatility.plugins.windows.cmdline*), 479

CMHIVE (*class in volatility.framework.symbols.windows.extensions.registry*), 361

CMHIVE.VolTemplateProxy (*class in volatility.framework.symbols.windows.extensions.registry*), 361

Column (*class in volatility.framework.interfaces.renderers*), 129

columns() (*TreeGrid property*), 130, 238

ColumnSortKey (*class in volatility.framework.interfaces.renderers*), 129

ColumnSortKey (*class in volatility.framework.renderers*), 237

CommandLine (*class in volatility.cli*), 28

ComplexListRequirement (*class in volatility.framework.configuration.requirements*), 74

config() (*AutomagicInterface property*), 99

config() (*Banners property*), 534

config() (*Bash property*), 409, 433

config() (*BashIntermedSymbols property*), 286

config() (*BigPools property*), 477

config() (*BufferDataLayer property*), 176

config() (*Check\_afinfo property*), 411

config() (*Check\_creds property*), 412

config() (*Check\_idt property*), 414

config() (*Check\_modules property*), 415

config() (*Check\_syscall property*), 417, 435

config() (*Check\_sysctl property*), 436

config() (*Check\_trap\_table property*), 438

config() (*CmdLine property*), 479

config() (*ConfigurableInterface property*), 103

config() (*ConfigWriter property*), 535

config() (*ConstructionMagic property*), 46

config() (*Context property*), 94

config() (*ContextInterface property*), 113

config() (*DataLayerInterface property*), 116

config() (*DllList property*), 481

config() (*DriverIrp property*), 483

config() (*DriverScan property*), 484

config() (*DumpFiles property*), 486

config() (*Elf64Layer property*), 147

config() (*Elfs property*), 418

config() (*Envars property*), 488

config() (*FileLayer property*), 179

config() (*FileScan property*), 490

config() (*FrameworkInfo property*), 537

config() (*GetServiceSIDs property*), 492

config() (*GetSIDs property*), 493

`config()` (*Handles property*), 495  
`config()` (*HiveList property*), 470  
`config()` (*HiveScan property*), 472  
`config()` (*Ifconfig property*), 439  
`config()` (*Info property*), 497  
`config()` (*Intel property*), 150  
`config()` (*Intel32e property*), 152  
`config()` (*IntelPAE property*), 154  
`config()` (*IntermediateSymbolTable property*), 381  
`config()` (*IsfInfo property*), 538  
`config()` (*ISFormatTable property*), 378  
`config()` (*Kauth\_listeners property*), 441  
`config()` (*Kauth\_scopes property*), 442  
`config()` (*KernelPDBScanner property*), 55  
`config()` (*Kevents property*), 444  
`config()` (*Keyboard\_notifiers property*), 420  
`config()` (*LayerStacker property*), 58  
`config()` (*LayerWriter property*), 540  
`config()` (*LimeLayer property*), 166  
`config()` (*LinearlyMappedLayer property*), 169  
`config()` (*LinuxBannerCache property*), 47  
`config()` (*LinuxKernelIntermedSymbols property*), 254  
`config()` (*LinuxSymbolFinder property*), 49  
`config()` (*List\_Files property*), 446  
`config()` (*Lsmmod property*), 421, 447  
`config()` (*Lsof property*), 423, 449  
`config()` (*MacBannerCache property*), 51  
`config()` (*MacKernelIntermedSymbols property*), 289  
`config()` (*MacSymbolFinder property*), 53  
`config()` (*Malfind property*), 425, 451, 499  
`config()` (*Maps property*), 426, 456  
`config()` (*Memmap property*), 501  
`config()` (*ModScan property*), 502  
`config()` (*Modules property*), 505  
`config()` (*Mount property*), 452  
`config()` (*MutantScan property*), 507  
`config()` (*NetScan property*), 509  
`config()` (*Netstat property*), 454  
`config()` (*NonLinearlySegmentedLayer property*), 188  
`config()` (*PdbMSFStream property*), 171  
`config()` (*PdbMultiStreamFormat property*), 174  
`config()` (*PluginInterface property*), 128  
`config()` (*PoolScanner property*), 512  
`config()` (*PrintKey property*), 474  
`config()` (*Privs property*), 514  
`config()` (*Psaux property*), 457  
`config()` (*PsList property*), 428, 459, 516  
`config()` (*PsScan property*), 518  
`config()` (*PsTree property*), 429, 462, 521  
`config()` (*QemuSuspendLayer property*), 182  
`config()` (*RegistryHive property*), 184  
`config()` (*SegmentedLayer property*), 190  
`config()` (*Socket\_filters property*), 463  
`config()` (*SSDT property*), 523  
`config()` (*Strings property*), 524  
`config()` (*SymbolBannerCache property*), 60  
`config()` (*SymbolFinder property*), 62  
`config()` (*SymbolTableInterface property*), 138  
`config()` (*SymlinkScan property*), 526  
`config()` (*Timeliner property*), 542  
`config()` (*Timers property*), 465  
`config()` (*TranslationLayerInterface property*), 120  
`config()` (*Trustedbsd property*), 466  
`config()` (*tty\_check property*), 431  
`config()` (*UserAssist property*), 476  
`config()` (*VadInfo property*), 528  
`config()` (*VerInfo property*), 530  
`config()` (*Version1Format property*), 384  
`config()` (*Version2Format property*), 386  
`config()` (*Version3Format property*), 389  
`config()` (*Version4Format property*), 391  
`config()` (*Version5Format property*), 394  
`config()` (*Version6Format property*), 397  
`config()` (*Version7Format property*), 399  
`config()` (*Version8Format property*), 402  
`config()` (*VFSevents property*), 468  
`config()` (*VirtMap property*), 532  
`config()` (*VmwareLayer property*), 193  
`config()` (*Volshell property*), 32, 34, 36, 39  
`config()` (*WindowsCrashDump32Layer property*), 141  
`config()` (*WindowsCrashDump64Layer property*), 144  
`config()` (*WindowsIntel property*), 157  
`config()` (*WindowsIntel32e property*), 159  
`config()` (*WindowsIntelPAE property*), 161  
`config()` (*WindowsKernelIntermedSymbols property*), 313  
`config()` (*WindowsMixin property*), 163  
`config()` (*WinSwapLayers property*), 66  
`config()` (*WintelHelper property*), 68  
`config_path()` (*AutomagicInterface property*), 99  
`config_path()` (*Banners property*), 534  
`config_path()` (*Bash property*), 409, 433  
`config_path()` (*BashIntermedSymbols property*), 286  
`config_path()` (*BigPools property*), 478  
`config_path()` (*BufferDataLayer property*), 177  
`config_path()` (*Check\_afinfo property*), 411  
`config_path()` (*Check\_creds property*), 412  
`config_path()` (*Check\_idt property*), 414  
`config_path()` (*Check\_modules property*), 415  
`config_path()` (*Check\_syscall property*), 417, 435  
`config_path()` (*Check\_sysctl property*), 436  
`config_path()` (*Check\_trap\_table property*), 438  
`config_path()` (*CmdLine property*), 479

`config_path()` (*ConfigurableInterface* property), 103

`config_path()` (*ConfigWriter* property), 535

`config_path()` (*ConstructionMagic* property), 46

`config_path()` (*DataLayerInterface* property), 116

`config_path()` (*DllList* property), 481

`config_path()` (*DriverIrp* property), 483

`config_path()` (*DriverScan* property), 484

`config_path()` (*DumpFiles* property), 486

`config_path()` (*Elf64Layer* property), 147

`config_path()` (*Elfs* property), 418

`config_path()` (*Envvars* property), 488

`config_path()` (*FileLayer* property), 179

`config_path()` (*FileScan* property), 490

`config_path()` (*FrameworkInfo* property), 537

`config_path()` (*GetServiceSIDs* property), 492

`config_path()` (*GetSIDs* property), 493

`config_path()` (*Handles* property), 495

`config_path()` (*HiveList* property), 470

`config_path()` (*HiveScan* property), 472

`config_path()` (*Ifconfig* property), 439

`config_path()` (*Info* property), 497

`config_path()` (*Intel* property), 150

`config_path()` (*Intel32e* property), 152

`config_path()` (*IntelPAE* property), 154

`config_path()` (*IntermediateSymbolTable* property), 381

`config_path()` (*IsfInfo* property), 538

`config_path()` (*ISFormatTable* property), 378

`config_path()` (*Kauth\_listeners* property), 441

`config_path()` (*Kauth\_scopes* property), 442

`config_path()` (*KernelPDBScanner* property), 55

`config_path()` (*Kevents* property), 444

`config_path()` (*Keyboard\_notifiers* property), 420

`config_path()` (*LayerStacker* property), 58

`config_path()` (*LayerWriter* property), 540

`config_path()` (*LimeLayer* property), 166

`config_path()` (*LinearlyMappedLayer* property), 169

`config_path()` (*LinuxBannerCache* property), 47

`config_path()` (*LinuxKernelIntermedSymbols* property), 254

`config_path()` (*LinuxSymbolFinder* property), 49

`config_path()` (*List\_Files* property), 446

`config_path()` (*Lsmode* property), 421, 448

`config_path()` (*Lsof* property), 423, 449

`config_path()` (*MacBannerCache* property), 51

`config_path()` (*MacKernelIntermedSymbols* property), 289

`config_path()` (*MacSymbolFinder* property), 53

`config_path()` (*Malfind* property), 425, 451, 499

`config_path()` (*Maps* property), 426, 456

`config_path()` (*Memmap* property), 501

`config_path()` (*ModScan* property), 502

`config_path()` (*Modules* property), 505

`config_path()` (*Mount* property), 452

`config_path()` (*MutantScan* property), 507

`config_path()` (*NetScan* property), 509

`config_path()` (*Netstat* property), 454

`config_path()` (*NonLinearlySegmentedLayer* property), 188

`config_path()` (*PdbMSFStream* property), 171

`config_path()` (*PdbMultiStreamFormat* property), 174

`config_path()` (*PluginInterface* property), 128

`config_path()` (*PoolScanner* property), 512

`config_path()` (*PrintKey* property), 474

`config_path()` (*Privs* property), 514

`config_path()` (*Psaux* property), 457

`config_path()` (*PsList* property), 428, 459, 516

`config_path()` (*PsScan* property), 519

`config_path()` (*PsTree* property), 430, 462, 521

`config_path()` (*QemuSuspendLayer* property), 182

`config_path()` (*RegistryHive* property), 184

`config_path()` (*SegmentedLayer* property), 190

`config_path()` (*Socket\_filters* property), 464

`config_path()` (*SSDT* property), 523

`config_path()` (*Strings* property), 524

`config_path()` (*SymbolBannerCache* property), 61

`config_path()` (*SymbolFinder* property), 62

`config_path()` (*SymbolTableInterface* property), 138

`config_path()` (*SymlinkScan* property), 526

`config_path()` (*Timeliner* property), 542

`config_path()` (*Timers* property), 465

`config_path()` (*TranslationLayerInterface* property), 120

`config_path()` (*Trustedbsd* property), 467

`config_path()` (*tty\_check* property), 432

`config_path()` (*UserAssist* property), 476

`config_path()` (*VadInfo* property), 528

`config_path()` (*VerInfo* property), 530

`config_path()` (*Version1Format* property), 384

`config_path()` (*Version2Format* property), 386

`config_path()` (*Version3Format* property), 389

`config_path()` (*Version4Format* property), 392

`config_path()` (*Version5Format* property), 394

`config_path()` (*Version6Format* property), 397

`config_path()` (*Version7Format* property), 399

`config_path()` (*Version8Format* property), 402

`config_path()` (*VFSevents* property), 468

`config_path()` (*VirtMap* property), 532

`config_path()` (*VmwareLayer* property), 193

`config_path()` (*Volshell* property), 32, 34, 36, 39

`config_path()` (*WindowsCrashDump32Layer* property), 141

`config_path()` (*WindowsCrashDump64Layer* property), 144

`config_path()` (*WindowsIntel* property), 157



- `config_path()` (*WindowsIntel32e property*), 159
- `config_path()` (*WindowsIntelPAE property*), 161
- `config_path()` (*WindowsKernelIntermedSymbols property*), 313
- `config_path()` (*WindowsMixin property*), 163
- `config_path()` (*WinSwapLayers property*), 66
- `config_path()` (*WintelHelper property*), 68
- `CONFIG_SEPARATOR` (in module *volatility.framework.interfaces.configuration*), 101
- `config_value()` (*BooleanRequirement method*), 70
- `config_value()` (*BytesRequirement method*), 71
- `config_value()` (*ChoiceRequirement method*), 73
- `config_value()` (*ClassRequirement method*), 102
- `config_value()` (*ComplexListRequirement method*), 74
- `config_value()` (*ConfigurableRequirementInterface method*), 104
- `config_value()` (*ConstructableRequirementInterface method*), 106
- `config_value()` (*IntRequirement method*), 76
- `config_value()` (*LayerListRequirement method*), 77
- `config_value()` (*ListRequirement method*), 79
- `config_value()` (*MultiRequirement method*), 81
- `config_value()` (*PluginRequirement method*), 82
- `config_value()` (*RequirementInterface method*), 109
- `config_value()` (*SimpleTypeRequirement method*), 111
- `config_value()` (*StringRequirement method*), 84
- `config_value()` (*SymbolTableRequirement method*), 85
- `config_value()` (*TranslationLayerRequirement method*), 87
- `config_value()` (*URIRequirement method*), 89
- `config_value()` (*VersionRequirement method*), 90
- `ConfigurableInterface` (class in *volatility.framework.interfaces.configuration*), 103
- `ConfigurableRequirementInterface` (class in *volatility.framework.interfaces.configuration*), 104
- `ConfigWriter` (class in *volatility.plugins.configwriter*), 535
- `conjugate()` (*Bin method*), 241
- `conjugate()` (*BitField method*), 199
- `conjugate()` (*Boolean method*), 201
- `conjugate()` (*Char method*), 208
- `conjugate()` (*Enumeration method*), 213
- `conjugate()` (*Float method*), 215
- `conjugate()` (*Hex method*), 242
- `conjugate()` (*Integer method*), 219
- `conjugate()` (*Pointer method*), 221
- `constant_data()` (*SymbolInterface property*), 136
- `construct()` (*ComplexListRequirement method*), 75
- `construct()` (*ConstructableRequirementInterface method*), 106
- `construct()` (*LayerListRequirement method*), 78
- `construct()` (*SymbolTableRequirement method*), 86
- `construct()` (*TranslationLayerRequirement method*), 88
- `construct_locals()` (*Volshell method*), 32, 34, 36, 39
- `construct_plugin()` (in module *volatility.framework.plugins*), 236
- `ConstructableRequirementInterface` (class in *volatility.framework.interfaces.configuration*), 105
- `ConstructionMagic` (class in *volatility.framework.automagic.construct\_layers*), 46
- `consume_padding()` (*PdbReader method*), 373
- `consume_type()` (*PdbReader method*), 374
- `Context` (class in *volatility.framework.contexts*), 93
- `context()` (*AutomagicInterface property*), 99
- `context()` (*Banners property*), 534
- `context()` (*Bash property*), 409, 433
- `context()` (*BashIntermedSymbols property*), 286
- `context()` (*BigPools property*), 478
- `context()` (*BufferDataLayer property*), 177
- `context()` (*BytesScanner property*), 140
- `context()` (*Check\_afinfo property*), 411
- `context()` (*Check\_creds property*), 412
- `context()` (*Check\_idt property*), 414
- `context()` (*Check\_modules property*), 415
- `context()` (*Check\_syscall property*), 417, 435
- `context()` (*Check\_sysctl property*), 437
- `context()` (*Check\_trap\_table property*), 438
- `context()` (*CmdLine property*), 479
- `context()` (*ConfigurableInterface property*), 103
- `context()` (*ConfigWriter property*), 535
- `context()` (*ConstructionMagic property*), 46
- `context()` (*DataLayerInterface property*), 116
- `context()` (*DllList property*), 481
- `context()` (*DriverIrp property*), 483
- `context()` (*DriverScan property*), 485
- `context()` (*DumpFiles property*), 486
- `context()` (*Elf64Layer property*), 147
- `context()` (*Elfs property*), 418
- `context()` (*Envvars property*), 488
- `context()` (*FileLayer property*), 179
- `context()` (*FileScan property*), 490
- `context()` (*FrameworkInfo property*), 537
- `context()` (*GetServiceSIDs property*), 492
- `context()` (*GetSIDs property*), 493
- `context()` (*Handles property*), 495
- `context()` (*HiveList property*), 470
- `context()` (*HiveScan property*), 472
- `context()` (*Ifconfig property*), 440

`context ()` (*Info property*), 497  
`context ()` (*Intel property*), 150  
`context ()` (*Intel32e property*), 152  
`context ()` (*IntelPAE property*), 155  
`context ()` (*IntermediateSymbolTable property*), 381  
`context ()` (*IsfInfo property*), 538  
`context ()` (*ISFormatTable property*), 378  
`context ()` (*Kauth\_listeners property*), 441  
`context ()` (*Kauth\_scopes property*), 443  
`context ()` (*KernelPDBScanner property*), 55  
`context ()` (*Kevents property*), 444  
`context ()` (*Keyboard\_notifiers property*), 420  
`context ()` (*LayerStacker property*), 58  
`context ()` (*LayerWriter property*), 540  
`context ()` (*LimeLayer property*), 166  
`context ()` (*LinearlyMappedLayer property*), 169  
`context ()` (*LinuxBannerCache property*), 48  
`context ()` (*LinuxKernelIntermedSymbols property*), 254  
`context ()` (*LinuxSymbolFinder property*), 50  
`context ()` (*List\_Files property*), 446  
`context ()` (*Lsmmod property*), 422, 448  
`context ()` (*Lsof property*), 423, 449  
`context ()` (*MacBannerCache property*), 51  
`context ()` (*MacKernelIntermedSymbols property*), 289  
`context ()` (*MacSymbolFinder property*), 53  
`context ()` (*Malfind property*), 425, 451, 499  
`context ()` (*Maps property*), 426, 456  
`context ()` (*Memmap property*), 501  
`context ()` (*ModScan property*), 503  
`context ()` (*Module property*), 95  
`context ()` (*ModuleInterface property*), 114  
`context ()` (*Modules property*), 505  
`context ()` (*Mount property*), 452  
`context ()` (*MultiStringScanner property*), 140  
`context ()` (*MutantScan property*), 507  
`context ()` (*NetScan property*), 509  
`context ()` (*Netstat property*), 454  
`context ()` (*NonLinearlySegmentedLayer property*), 188  
`context ()` (*PageMapScanner property*), 66  
`context ()` (*PdbMSFStream property*), 172  
`context ()` (*PdbMultiStreamFormat property*), 174  
`context ()` (*PdbReader property*), 374  
`context ()` (*PdbSignatureScanner property*), 376  
`context ()` (*PluginInterface property*), 128  
`context ()` (*PoolHeaderScanner property*), 511  
`context ()` (*PoolScanner property*), 512  
`context ()` (*PrintKey property*), 474  
`context ()` (*Privs property*), 514  
`context ()` (*Psaux property*), 457  
`context ()` (*PsList property*), 428, 459, 516  
`context ()` (*PsScan property*), 519  
`context ()` (*PsTree property*), 430, 462, 521  
`context ()` (*QemuSuspendLayer property*), 182  
`context ()` (*RegExScanner property*), 140  
`context ()` (*RegistryHive property*), 184  
`context ()` (*ScannerInterface property*), 120  
`context ()` (*SegmentedLayer property*), 191  
`context ()` (*SizedModule property*), 97  
`context ()` (*Socket\_filters property*), 464  
`context ()` (*SSDT property*), 523  
`context ()` (*Strings property*), 524  
`context ()` (*SymbolBannerCache property*), 61  
`context ()` (*SymbolFinder property*), 62  
`context ()` (*SymbolTableInterface property*), 138  
`context ()` (*SymlinkScan property*), 526  
`context ()` (*Timeliner property*), 542  
`context ()` (*Timers property*), 465  
`context ()` (*TranslationLayerInterface property*), 120  
`context ()` (*Trustedbsd property*), 467  
`context ()` (*tty\_check property*), 432  
`context ()` (*UserAssist property*), 476  
`context ()` (*VadInfo property*), 528  
`context ()` (*VerInfo property*), 530  
`context ()` (*Version1Format property*), 384  
`context ()` (*Version2Format property*), 386  
`context ()` (*Version3Format property*), 389  
`context ()` (*Version4Format property*), 392  
`context ()` (*Version5Format property*), 394  
`context ()` (*Version6Format property*), 397  
`context ()` (*Version7Format property*), 399  
`context ()` (*Version8Format property*), 402  
`context ()` (*VFSevents property*), 468  
`context ()` (*VirtMap property*), 532  
`context ()` (*VmwareLayer property*), 193  
`context ()` (*Volshell property*), 32, 34, 36, 39  
`context ()` (*WindowsCrashDump32Layer property*), 142  
`context ()` (*WindowsCrashDump64Layer property*), 144  
`context ()` (*WindowsIntel property*), 157  
`context ()` (*WindowsIntel32e property*), 159  
`context ()` (*WindowsIntelPAE property*), 161  
`context ()` (*WindowsKernelIntermedSymbols property*), 313  
`context ()` (*WindowsMixin property*), 164  
`context ()` (*WinSwapLayers property*), 67  
`context ()` (*WintelHelper property*), 68  
`ContextInterface` (class in `volatility.framework.interfaces.context`), 112  
`CONTROL_AREA` (class in `volatility.framework.symbols.windows.extensions`), 315  
`CONTROL_AREA.VolTemplateProxy` (class in `volatility.framework.symbols.windows.extensions`),

315  
 convert\_arg\_line\_to\_args() (*HelpfulArg-Parser method*), 43  
 convert\_bytes\_to\_guid() (*PdbReader method*), 374  
 convert\_data\_to\_value() (*in module volatility.framework.objects*), 234  
 convert\_fields() (*PdbReader method*), 374  
 convert\_ipv4() (*in module volatility.framework.renderers.conversion*), 240  
 convert\_ipv6() (*in module volatility.framework.renderers.conversion*), 240  
 convert\_network\_four\_tuple() (*in module volatility.framework.renderers.conversion*), 240  
 convert\_port() (*in module volatility.framework.renderers.conversion*), 240  
 convert\_value\_to\_data() (*in module volatility.framework.objects*), 234  
 count() (*Array property*), 198  
 count() (*Bytes method*), 204  
 count() (*Column method*), 129  
 count() (*DataFormatInfo method*), 211  
 count() (*HexBytes method*), 243  
 count() (*MultiTypeData method*), 246  
 count() (*String method*), 225  
 count() (*TreeNode method*), 132, 239  
 crashdump\_json (*WindowsCrashDump32Layer attribute*), 142  
 crashdump\_json (*WindowsCrashDump64Layer attribute*), 144  
 create() (*BashIntermedSymbols class method*), 286  
 create() (*IntermediateSymbolTable class method*), 381  
 create() (*LinuxKernelIntermedSymbols class method*), 254  
 create() (*MacKernelIntermedSymbols class method*), 289  
 create() (*WindowsKernelIntermedSymbols class method*), 313  
 create\_configurable() (*Volshell method*), 32, 34, 37, 39  
 create\_json\_hash() (*in module volatility.schemas*), 543  
 create\_name\_filter() (*PsList class method*), 516  
 create\_netscan\_constraints() (*NetScan static method*), 509  
 create\_netscan\_symbol\_table() (*NetScan class method*), 509  
 create\_pid\_filter() (*PsList class method*), 428, 459, 516  
 create\_pid\_filter() (*PsTree class method*), 430  
 create\_stackers\_list() (*LayerStacker method*), 58

create\_stream\_from\_pages() (*PdbMulti-StreamFormat method*), 174  
 CREATED (*TimeLinerType attribute*), 542  
 createservicesid() (*in module volatility.plugins.windows.getservicesids*), 493  
 CSVRenderer (*class in volatility.cli.text\_renderer*), 41  
 current\_layer() (*Volshell property*), 32, 34, 37, 39

## D

data() (*HierarchicalDict property*), 108  
 DataFormatInfo (*class in volatility.framework.objects*), 211  
 DataLayerInterface (*class in volatility.framework.interfaces.layers*), 116  
 decode() (*Bytes method*), 204  
 decode() (*HexBytes method*), 243  
 decode() (*MultiTypeData method*), 246  
 decode\_data() (*CM\_KEY\_VALUE method*), 366  
 deduplicate() (*ModuleCollection method*), 96  
 default() (*BooleanRequirement property*), 70  
 default() (*BytesRequirement property*), 72  
 default() (*ChoiceRequirement property*), 73  
 default() (*ClassRequirement property*), 102  
 default() (*ComplexListRequirement property*), 75  
 default() (*ConfigurableRequirementInterface property*), 104  
 default() (*ConstructableRequirementInterface property*), 106  
 default() (*IntRequirement property*), 76  
 default() (*LayerListRequirement property*), 78  
 default() (*ListRequirement property*), 80  
 default() (*MultiRequirement property*), 81  
 default() (*PluginRequirement property*), 83  
 default() (*RequirementInterface property*), 109  
 default() (*SimpleTypeRequirement property*), 111  
 default() (*StringRequirement property*), 84  
 default() (*SymbolTableRequirement property*), 86  
 default() (*TranslationLayerRequirement property*), 88  
 default() (*URIRequirement property*), 89  
 default() (*VersionRequirement property*), 90  
 default\_block\_size (*LayerWriter attribute*), 540  
 default\_open() (*JarHandler static method*), 187  
 del\_layer() (*LayerContainer method*), 119  
 del\_type\_class() (*BaseSymbolTableInterface method*), 133  
 del\_type\_class() (*BashIntermedSymbols method*), 286  
 del\_type\_class() (*IntermediateSymbolTable method*), 382  
 del\_type\_class() (*ISFormatTable method*), 378  
 del\_type\_class() (*LinuxKernelIntermedSymbols method*), 254

`del_type_class()` (*MacKernelIntermedSymbols method*), 289

`del_type_class()` (*NativeTable method*), 405

`del_type_class()` (*NativeTableInterface method*), 134

`del_type_class()` (*SymbolTableInterface method*), 138

`del_type_class()` (*Version1Format method*), 384

`del_type_class()` (*Version2Format method*), 387

`del_type_class()` (*Version3Format method*), 389

`del_type_class()` (*Version4Format method*), 392

`del_type_class()` (*Version5Format method*), 394

`del_type_class()` (*Version6Format method*), 397

`del_type_class()` (*Version7Format method*), 400

`del_type_class()` (*Version8Format method*), 402

`del_type_class()` (*WindowsKernelIntermedSymbols method*), 313

`deleter()` (*classproperty method*), 27

`denominator` (*Bin attribute*), 241

`denominator` (*BitField attribute*), 199

`denominator` (*Boolean attribute*), 202

`denominator` (*Char attribute*), 208

`denominator` (*Enumeration attribute*), 213

`denominator` (*Hex attribute*), 242

`denominator` (*Integer attribute*), 219

`denominator` (*Pointer attribute*), 221

`dentry` (*class in volatility.framework.symbols.linux.extensions*), 257

`dentry.VolTemplateProxy` (*class in volatility.framework.symbols.linux.extensions*), 257

`dependencies()` (*BufferDataLayer property*), 177

`dependencies()` (*DataLayerInterface property*), 116

`dependencies()` (*Elf64Layer property*), 147

`dependencies()` (*FileLayer property*), 179

`dependencies()` (*Intel property*), 150

`dependencies()` (*Intel32e property*), 152

`dependencies()` (*IntelPAE property*), 155

`dependencies()` (*LimeLayer property*), 166

`dependencies()` (*LinearlyMappedLayer property*), 169

`dependencies()` (*NonLinearlySegmentedLayer property*), 188

`dependencies()` (*PdbMSFStream property*), 172

`dependencies()` (*PdbMultiStreamFormat property*), 174

`dependencies()` (*QemuSuspendLayer property*), 182

`dependencies()` (*RegistryHive property*), 184

`dependencies()` (*SegmentedLayer property*), 191

`dependencies()` (*TranslationLayerInterface property*), 120

`dependencies()` (*VmwareLayer property*), 193

`dependencies()` (*WindowsCrashDump32Layer property*), 142

`dependencies()` (*WindowsCrashDump64Layer property*), 144

`dependencies()` (*WindowsIntel property*), 157

`dependencies()` (*WindowsIntel32e property*), 159

`dependencies()` (*WindowsIntelPAE property*), 161

`dependencies()` (*WindowsMixin property*), 164

`dereference()` (*EX\_FAST\_REF method*), 325

`dereference()` (*Pointer method*), 221

`description()` (*BooleanRequirement property*), 70

`description()` (*BytesRequirement property*), 72

`description()` (*ChoiceRequirement property*), 73

`description()` (*ClassRequirement property*), 102

`description()` (*ComplexListRequirement property*), 75

`description()` (*ConfigurableRequirementInterface property*), 105

`description()` (*ConstructableRequirementInterface property*), 106

`description()` (*Enumeration property*), 213

`description()` (*IntRequirement property*), 76

`description()` (*LayerListRequirement property*), 78

`description()` (*ListRequirement property*), 80

`description()` (*MultiRequirement property*), 81

`description()` (*PluginRequirement property*), 83

`description()` (*RequirementInterface property*), 109

`description()` (*SimpleTypeRequirement property*), 111

`description()` (*StringRequirement property*), 84

`description()` (*SymbolTableRequirement property*), 86

`description()` (*TranslationLayerRequirement property*), 88

`description()` (*URIRequirement property*), 89

`description()` (*VersionRequirement property*), 90

`destroy()` (*BufferDataLayer method*), 177

`destroy()` (*DataLayerInterface method*), 116

`destroy()` (*Elf64Layer method*), 147

`destroy()` (*FileLayer method*), 179

`destroy()` (*Intel method*), 150

`destroy()` (*Intel32e method*), 152

`destroy()` (*IntelPAE method*), 155

`destroy()` (*LimeLayer method*), 166

`destroy()` (*LinearlyMappedLayer method*), 169

`destroy()` (*NonLinearlySegmentedLayer method*), 189

`destroy()` (*PdbMSFStream method*), 172

`destroy()` (*PdbMultiStreamFormat method*), 174

`destroy()` (*QemuSuspendLayer method*), 182

`destroy()` (*RegistryHive method*), 185

`destroy()` (*SegmentedLayer method*), 191

`destroy()` (*TranslationLayerInterface method*), 120

`destroy()` (*VmwareLayer method*), 193

`destroy()` (*WindowsCrashDump32Layer method*), 142



[destroy\(\) \(WindowsCrashDump64Layer method\), 144](#)  
[destroy\(\) \(WindowsIntel method\), 157](#)  
[destroy\(\) \(WindowsIntel32e method\), 159](#)  
[destroy\(\) \(WindowsIntelPAE method\), 161](#)  
[destroy\(\) \(WindowsMixin method\), 164](#)  
[detach\(\) \(NullFileHandler method\), 30](#)  
[determine\\_extended\\_value\(\) \(PdbReader method\), 374](#)  
[determine\\_map\(\) \(VirtMap class method\), 532](#)  
[determine\\_tcpip\\_version\(\) \(NetScan class method\), 509](#)  
[determine\\_valid\\_kernel\(\) \(KernelPDBScanner method\), 55](#)  
[DEVICE\\_OBJECT \(class in volatility.framework.symbols.windows.extensions\), 317](#)  
[DEVICE\\_OBJECT.VolTemplateProxy \(class in volatility.framework.symbols.windows.extensions\), 317](#)  
[disassemble\(\) \(Volshell method\), 32, 34, 37, 39](#)  
[Disassembly \(class in volatility.framework.interfaces.renderers\), 130](#)  
[display\\_bytes\(\) \(Volshell method\), 32, 34, 37, 39](#)  
[display\\_disassembly\(\) \(in module volatility.cli.text\\_renderer\), 42](#)  
[display\\_doublewords\(\) \(Volshell method\), 32, 34, 37, 39](#)  
[display\\_plugin\\_output\(\) \(Volshell method\), 32, 34, 37, 39](#)  
[display\\_quadwords\(\) \(Volshell method\), 32, 35, 37, 39](#)  
[display\\_symbols\(\) \(Volshell method\), 32, 35, 37, 39](#)  
[display\\_type\(\) \(Volshell method\), 32, 35, 37, 39](#)  
[display\\_words\(\) \(Volshell method\), 32, 35, 37, 39](#)  
[DllList \(class in volatility.plugins.windows.dlllist\), 481](#)  
[download\\_pdb\\_isf\(\) \(PDBUtility class method\), 375](#)  
[DRIVER\\_OBJECT \(class in volatility.framework.symbols.windows.extensions\), 318](#)  
[DRIVER\\_OBJECT.VolTemplateProxy \(class in volatility.framework.symbols.windows.extensions\), 319](#)  
[DriverIrp \(class in volatility.plugins.windows.driverirp\), 483](#)  
[DriverScan \(class in volatility.plugins.windows.driverscan\), 484](#)  
[DtbSelfRef32bit \(class in volatility.framework.automagic.windows\), 64](#)  
[DtbSelfRef64bit \(class in volatility.framework.automagic.windows\), 64](#)  
[DtbSelfReferential \(class in volatility.framework.automagic.windows\), 64](#)  
[DtbTest \(class in volatility.framework.automagic.windows\), 65](#)  
[DtbTest32bit \(class in volatility.framework.automagic.windows\), 65](#)  
[DtbTest64bit \(class in volatility.framework.automagic.windows\), 65](#)  
[DtbTestPae \(class in volatility.framework.automagic.windows\), 66](#)  
[DummyLock \(class in volatility.framework.layers.physical\), 178](#)  
[DummyProgress \(class in volatility.framework.interfaces.layers\), 118](#)  
[dump\\_file\\_producer\(\) \(DumpFiles class method\), 486](#)  
[dump\\_header\\_name \(WindowsCrashDump32Layer attribute\), 142](#)  
[dump\\_header\\_name \(WindowsCrashDump64Layer attribute\), 144](#)  
[dump\\_pe\(\) \(DllList class method\), 481](#)  
[DumpFiles \(class in volatility.plugins.windows.dumpfiles\), 486](#)  
[dynamic\\_sections\(\) \(elf\\_phdr method\), 283](#)

## E

[elf \(class in volatility.framework.symbols.linux.extensions.elf\), 280](#)  
[elf.VolTemplateProxy \(class in volatility.framework.symbols.linux.extensions.elf\), 280](#)  
[Elf64Layer \(class in volatility.framework.layers.elf\), 147](#)  
[Elf64Stacker \(class in volatility.framework.layers.elf\), 149](#)  
[ELF\\_CLASS \(Elf64Layer attribute\), 147](#)  
[elf\\_phdr \(class in volatility.framework.symbols.linux.extensions.elf\), 282](#)  
[elf\\_phdr.VolTemplateProxy \(class in volatility.framework.symbols.linux.extensions.elf\), 282](#)  
[elf\\_sym \(class in volatility.framework.symbols.linux.extensions.elf\), 283](#)  
[elf\\_sym.VolTemplateProxy \(class in volatility.framework.symbols.linux.extensions.elf\), 284](#)  
[ElfFormatException, 149](#)  
[Elfs \(class in volatility.plugins.linux.elfs\), 418](#)  
[encode\(\) \(String method\), 225](#)

`endswith()` (*Bytes method*), 204  
`endswith()` (*HexBytes method*), 243  
`endswith()` (*MultiTypeData method*), 246  
`endswith()` (*String method*), 225  
`ENUM` (*SymbolType attribute*), 251  
`Enumeration` (*class in volatility.framework.objects*), 212  
`Enumeration.VolTemplateProxy` (*class in volatility.framework.objects*), 212  
`enumerations()` (*BaseSymbolTableInterface property*), 133  
`enumerations()` (*BashIntermedSymbols property*), 286  
`enumerations()` (*IntermediateSymbolTable property*), 382  
`enumerations()` (*ISFormatTable property*), 378  
`enumerations()` (*LinuxKernelIntermedSymbols property*), 254  
`enumerations()` (*MacKernelIntermedSymbols property*), 289  
`enumerations()` (*NativeTable property*), 405  
`enumerations()` (*NativeTableInterface property*), 134  
`enumerations()` (*SymbolTableInterface property*), 138  
`enumerations()` (*Version1Format property*), 384  
`enumerations()` (*Version2Format property*), 387  
`enumerations()` (*Version3Format property*), 389  
`enumerations()` (*Version4Format property*), 392  
`enumerations()` (*Version5Format property*), 394  
`enumerations()` (*Version6Format property*), 397  
`enumerations()` (*Version7Format property*), 400  
`enumerations()` (*Version8Format property*), 402  
`enumerations()` (*WindowsKernelIntermedSymbols property*), 313  
`Envvars` (*class in volatility.plugins.windows.envvars*), 488  
`environment variable`  
    PYTHONPATH, 25  
`environment_variables()` (*EPROCESS method*), 321  
`EPROCESS` (*class in volatility.framework.symbols.windows.extensions*), 320  
`EPROCESS.VolTemplateProxy` (*class in volatility.framework.symbols.windows.extensions*), 320  
`error()` (*HelpfulArgParser method*), 43  
`ETHREAD` (*class in volatility.framework.symbols.windows.extensions*), 322  
`ETHREAD.VolTemplateProxy` (*class in volatility.framework.symbols.windows.extensions*), 323  
`event_types` (*Kevents attribute*), 444  
`event_types` (*VFSevents attribute*), 468  
`EX_FAST_REF` (*class in volatility.framework.symbols.windows.extensions*), 324  
`EX_FAST_REF.VolTemplateProxy` (*class in volatility.framework.symbols.windows.extensions*), 324  
`exclusion_list` (*Elf64Stacker attribute*), 149  
`exclusion_list` (*LimeStacker attribute*), 168  
`exclusion_list` (*LinuxIntelStacker attribute*), 49  
`exclusion_list` (*MacIntelStacker attribute*), 52  
`exclusion_list` (*QemuStacker attribute*), 181  
`exclusion_list` (*StackerLayerInterface attribute*), 101  
`exclusion_list` (*VmwareStacker attribute*), 195  
`exclusion_list` (*WindowsCrashDumpStacker attribute*), 146  
`exclusion_list` (*WindowsIntelStacker attribute*), 68  
`ExecutiveObject` (*class in volatility.framework.symbols.windows.extensions.pool*), 352  
`ExecutiveObject.VolTemplateProxy` (*class in volatility.framework.symbols.windows.extensions.pool*), 352  
`exit()` (*HelpfulArgParser method*), 43  
`expandtabs()` (*Bytes method*), 204  
`expandtabs()` (*HexBytes method*), 243  
`expandtabs()` (*MultiTypeData method*), 246  
`expandtabs()` (*String method*), 225  
`extended_flags` (*vm\_area\_struct attribute*), 277  
`extract_data()` (*QemuSuspendLayer method*), 182

## F

`fdel` (*classproperty attribute*), 27  
`fget` (*classproperty attribute*), 27  
`file_handler_class_factory()` (*Command-Line method*), 28  
`file_handler_class_factory()` (*VolShell method*), 29  
`file_name_with_device()` (*FILE\_OBJECT method*), 326  
`FILE_OBJECT` (*class in volatility.framework.symbols.windows.extensions*), 325  
`FILE_OBJECT.VolTemplateProxy` (*class in volatility.framework.symbols.windows.extensions*), 326  
`file_symbol_url()` (*BashIntermedSymbols class method*), 286  
`file_symbol_url()` (*IntermediateSymbolTable class method*), 382

- `file_symbol_url()` (*LinuxKernelIntermedSymbols class method*), 255
- `file_symbol_url()` (*MacKernelIntermedSymbols class method*), 289
- `file_symbol_url()` (*WindowsKernelIntermedSymbols class method*), 313
- `fileglob` (class in *volatility.framework.symbols.mac.extensions*), 292
- `fileglob.VolTemplateProxy` (class in *volatility.framework.symbols.mac.extensions*), 292
- `FileHandlerInterface` (class in *volatility.framework.interfaces.plugins*), 126
- `FileLayer` (class in *volatility.framework.layers.physical*), 178
- `fileno()` (*FileHandlerInterface method*), 126
- `fileno()` (*NullFileHandler method*), 30
- `FILEOFFSET_MASK` (*VACB attribute*), 345
- `files_descriptors_for_process()` (*LinuxUtilities class method*), 256
- `files_descriptors_for_process()` (*MacUtilities class method*), 291
- `files_struct` (class in *volatility.framework.symbols.linux.extensions*), 258
- `files_struct.VolTemplateProxy` (class in *volatility.framework.symbols.linux.extensions*), 258
- `FileScan` (class in *volatility.plugins.windows.filescan*), 490
- `find()` (*Bytes method*), 204
- `find()` (*HexBytes method*), 243
- `find()` (*MultiTypeData method*), 246
- `find()` (*String method*), 225
- `find_aslr` (*SymbolFinder attribute*), 62
- `find_aslr()` (*LinuxIntelStacker class method*), 49
- `find_aslr()` (*LinuxSymbolFinder method*), 50
- `find_aslr()` (*MacIntelStacker class method*), 52
- `find_aslr()` (*MacSymbolFinder class method*), 53
- `find_cookie()` (*Handles class method*), 495
- `find_level()` (*PsTree method*), 430, 521
- `find_module()` (*WarningFindSpec method*), 27
- `find_requirements()` (*AutomagicInterface method*), 100
- `find_requirements()` (*ConstructionMagic method*), 46
- `find_requirements()` (*KernelPDBScanner method*), 55
- `find_requirements()` (*LayerStacker method*), 58
- `find_requirements()` (*LinuxBannerCache method*), 48
- `find_requirements()` (*LinuxSymbolFinder method*), 50
- `find_requirements()` (*MacBannerCache method*), 51
- `find_requirements()` (*MacSymbolFinder method*), 53
- `find_requirements()` (*SymbolBannerCache method*), 61
- `find_requirements()` (*SymbolFinder method*), 62
- `find_requirements()` (*WinSwapLayers method*), 67
- `find_requirements()` (*WintelHelper method*), 68
- `find_sar_value()` (*Handles method*), 495
- `find_session_layer()` (*ModScan class method*), 503
- `find_session_layer()` (*Modules class method*), 505
- `find_sid_re()` (in *module volatility.plugins.windows.getsids*), 494
- `find_spec()` (*WarningFindSpec static method*), 27
- `find_suitable_requirements()` (*LayerStacker class method*), 59
- `find_swap_requirement()` (*WinSwapLayers static method*), 67
- `find_virtual_layers_from_req()` (*KernelPDBScanner method*), 56
- `fix_image_base()` (*IMAGE\_DOS\_HEADER method*), 348
- `Flags` (class in *volatility.framework.symbols.wrappers*), 406
- `Float` (class in *volatility.framework.objects*), 214
- `Float.VolTemplateProxy` (class in *volatility.framework.objects*), 214
- `flush()` (*FileHandlerInterface method*), 126
- `flush()` (*NullFileHandler method*), 30
- `format()` (*String method*), 225
- `format_help()` (*HelpfulArgParser method*), 43
- `format_map()` (*String method*), 226
- `format_mapping` (*Version4Format attribute*), 392
- `format_mapping` (*Version5Format attribute*), 394
- `format_mapping` (*Version6Format attribute*), 397
- `format_mapping` (*Version7Format attribute*), 400
- `format_mapping` (*Version8Format attribute*), 402
- `format_usage()` (*HelpfulArgParser method*), 43
- `ForwardArrayCount` (class in *volatility.framework.symbols.windows.pdbconv*), 373
- `FrameworkInfo` (class in *volatility.plugins.frameworkinfo*), 536
- `FREE` (*PoolType attribute*), 514
- `free_layer_name()` (*LayerContainer method*), 119
- `free_table_name()` (*SymbolSpace method*), 250
- `free_table_name()` (*SymbolSpaceInterface method*), 136
- `from_bytes()` (*Bin method*), 241
- `from_bytes()` (*BitField method*), 199
- `from_bytes()` (*Boolean method*), 202
- `from_bytes()` (*Char method*), 208

`from_bytes()` (*Enumeration method*), 213  
`from_bytes()` (*Hex method*), 242  
`from_bytes()` (*Integer method*), 219  
`from_bytes()` (*Pointer method*), 221  
`fromhex()` (*Bytes method*), 204  
`fromhex()` (*Float method*), 215  
`fromhex()` (*HexBytes method*), 243  
`fromhex()` (*MultiTypeData method*), 246  
`fs_struct` (class in *volatility.framework.symbols.linux.extensions*), 260  
`fs_struct.VolTemplateProxy` (class in *volatility.framework.symbols.linux.extensions*), 260  
`fset` (classproperty attribute), 27  
`full_path()` (*vnnode method*), 311  
`Function` (class in *volatility.framework.objects*), 216  
`Function.VolTemplateProxy` (class in *volatility.framework.objects*), 216

## G

`generate_kernel_handler_info()` (*LinuxUtilities class method*), 256  
`generate_kernel_handler_info()` (*MacUtilities class method*), 291  
`generate_mapping()` (*Strings method*), 524  
`generate_pool_scan()` (*PoolScanner class method*), 512  
`generate_timeline()` (*Bash method*), 409, 433  
`generate_timeline()` (*DllList method*), 481  
`generate_timeline()` (*NetScan method*), 510  
`generate_timeline()` (*PsList method*), 516  
`generate_timeline()` (*PsScan method*), 519  
`generate_timeline()` (*SymlinkScan method*), 526  
`generate_timeline()` (*TimeLinerInterface method*), 542  
`generate_treegrid()` (*Volshell method*), 32, 35, 37, 39  
`generator()` (*HierarchicalDict method*), 108  
`GenericIntelProcess` (class in *volatility.framework.symbols.generic*), 252  
`GenericIntelProcess.VolTemplateProxy` (class in *volatility.framework.symbols.generic*), 252  
`get()` (*HierarchicalDict method*), 108  
`get()` (*LayerContainer method*), 119  
`get()` (*ObjectInformation method*), 123  
`get()` (*ReadOnlyMapping method*), 125  
`get()` (*RegValueTypes class method*), 369  
`get()` (*SymbolSpace method*), 250  
`get()` (*SymbolSpaceInterface method*), 136  
`get_address()` (*sockaddr method*), 302  
`get_available_pages()` (*CONTROL\_AREA method*), 316

`get_available_pages()` (*SHARED\_CACHE\_MAP method*), 340  
`get_binary()` (*SERVICE\_RECORD method*), 372  
`get_block_offset()` (*HMAP\_ENTRY method*), 368  
`get_build_lab()` (*KDDEBUGGER\_DATA64 method*), 347  
`get_cell()` (*RegistryHive method*), 185  
`get_cmdline()` (*CmdLine class method*), 479  
`get_command()` (*hist\_entry method*), 279  
`get_commit_charge()` (*MMVAD method*), 334  
`get_commit_charge()` (*MMVAD\_SHORT method*), 336  
`get_connection_info()` (*socket method*), 305  
`get_converted_connection_info()` (*socket method*), 305  
`get_core_size()` (*module method*), 266  
`get_create_time()` (*EPROCESS method*), 321  
`get_create_time()` (*OBJECT\_SYMBOLIC\_LINK method*), 338  
`get_cross_thread_flags()` (*ETHREAD method*), 323  
`get_csdversion()` (*KDDEBUGGER\_DATA64 method*), 347  
`get_ctltype()` (*sysctl\_oid method*), 306  
`get_default()` (*HelpfulArgParser method*), 43  
`get_dentry()` (*struct\_file method*), 271  
`get_depends()` (*Info class method*), 497  
`get_device_name()` (*DEVICE\_OBJECT method*), 318  
`get_display()` (*SERVICE\_RECORD method*), 372  
`get_driver_name()` (*DRIVER\_OBJECT method*), 319  
`get_end()` (*MMVAD method*), 334  
`get_end()` (*MMVAD\_SHORT method*), 336  
`get_enumeration()` (*BashIntermedSymbols method*), 287  
`get_enumeration()` (*IntermediateSymbolTable method*), 382  
`get_enumeration()` (*LinuxKernelIntermedSymbols method*), 255  
`get_enumeration()` (*MacKernelIntermedSymbols method*), 290  
`get_enumeration()` (*Module method*), 95  
`get_enumeration()` (*ModuleInterface method*), 114  
`get_enumeration()` (*NativeTable method*), 405  
`get_enumeration()` (*NativeTableInterface method*), 134  
`get_enumeration()` (*SizedModule method*), 97  
`get_enumeration()` (*SymbolSpace method*), 250  
`get_enumeration()` (*SymbolSpaceInterface method*), 136  
`get_enumeration()` (*Version1Format method*), 384  
`get_enumeration()` (*Version2Format method*), 387



`get_enumeration()` (*Version3Format method*), 389  
`get_enumeration()` (*Version4Format method*), 392  
`get_enumeration()` (*Version5Format method*), 395  
`get_enumeration()` (*Version6Format method*), 397  
`get_enumeration()` (*Version7Format method*), 400  
`get_enumeration()` (*Version8Format method*), 402  
`get_enumeration()` (*WindowsKernelIntermedSymbols method*), 314  
`get_exit_time()` (*EPROCESS method*), 321  
`get_family()` (*socket method*), 305  
`get_fds()` (*files\_struct method*), 259  
`get_fg_type()` (*fileglob method*), 293  
`get_file_name()` (*MMVAD method*), 334  
`get_file_name()` (*MMVAD\_SHORT method*), 336  
`get_file_offset()` (*VACB method*), 345  
`get_flags()` (*vm\_area\_struct method*), 277  
`get_full_key_name()` (*CM\_KEY\_BODY method*), 363  
`get_guid_from_mz()` (*PDBUtility class method*), 375  
`get_handle_count()` (*EPROCESS method*), 321  
`get_init_size()` (*module method*), 266  
`get_inpcb()` (*socket method*), 305  
`get_ipv4_info()` (*inpcb method*), 295  
`get_ipv6_info()` (*inpcb method*), 295  
`get_is_wow64()` (*EPROCESS method*), 321  
`get_json()` (*PdbReader method*), 374  
`get_kdbg_structure()` (*Info class method*), 497  
`get_kernel_module()` (*Info class method*), 497  
`get_key()` (*POOL\_TRACKER\_BIG\_PAGES method*), 359  
`get_key()` (*RegistryHive method*), 185  
`get_key_path()` (*CM\_KEY\_NODE method*), 365  
`get_kset_modules()` (*Check\_modules method*), 415  
`get_kuser_structure()` (*Info class method*), 497  
`get_left_child()` (*MMVAD method*), 334  
`get_left_child()` (*MMVAD\_SHORT method*), 336  
`get_link_name()` (*OBJECT\_SYMBOLIC\_LINK method*), 338  
`get_list_tasks()` (*PsList class method*), 459  
`get_listeners()` (*kauth\_scope method*), 297  
`get_map_iter()` (*proc method*), 298  
`get_map_object()` (*vm\_map\_object method*), 310  
`get_max_fds()` (*files\_struct method*), 259  
`get_mmap_iter()` (*mm\_struct method*), 265  
`get_mnt_flags()` (*mount method*), 268  
`get_mnt_mountpoint()` (*mount method*), 268  
`get_mnt_mountpoint()` (*vfsmount method*), 276  
`get_mnt_parent()` (*mount method*), 268  
`get_mnt_parent()` (*vfsmount method*), 276  
`get_mnt_root()` (*mount method*), 268  
`get_mnt_root()` (*vfsmount method*), 276  
`get_mnt_sb()` (*mount method*), 268  
`get_module_base()` (*module method*), 266  
`get_module_core()` (*module method*), 266  
`get_module_init()` (*module method*), 266  
`get_module_symbols_by_absolute_location()` (*ModuleCollection method*), 96  
`get_module_wrapper()` (*in module volatility.framework.contexts*), 98  
`get_name()` (*CM\_KEY\_NODE method*), 365  
`get_name()` (*CM\_KEY\_VALUE method*), 366  
`get_name()` (*CMHIVE method*), 361  
`get_name()` (*elf\_sym method*), 284  
`get_name()` (*KMUTANT method*), 328  
`get_name()` (*module method*), 266  
`get_name()` (*RegistryHive method*), 185  
`get_name()` (*SERVICE\_RECORD method*), 372  
`get_name()` (*vm\_area\_struct method*), 277  
`get_node()` (*RegistryHive method*), 185  
`get_nt_header()` (*IMAGE\_DOS\_HEADER method*), 349  
`get_nthheader_structure()` (*Info class method*), 497  
`get_number_of_bytes()` (*POOL\_TRACKER\_BIG\_PAGES method*), 359  
`get_object()` (*POOL\_HEADER method*), 356  
`get_object()` (*POOL\_HEADER\_VISTA method*), 358  
`get_object()` (*vm\_map\_entry method*), 308  
`get_object_header()` (*DEVICE\_OBJECT method*), 318  
`get_object_header()` (*DRIVER\_OBJECT method*), 319  
`get_object_header()` (*EPROCESS method*), 321  
`get_object_header()` (*ExecutiveObject method*), 352  
`get_object_header()` (*FILE\_OBJECT method*), 326  
`get_object_header()` (*KMUTANT method*), 328  
`get_object_header()` (*OBJECT\_SYMBOLIC\_LINK method*), 338  
`get_object_type()` (*OBJECT\_HEADER method*), 354  
`get_offset()` (*vm\_map\_entry method*), 308  
`get_osversion()` (*PsScan class method*), 519  
`get_page_offset()` (*vm\_area\_struct method*), 277  
`get_parent()` (*MMVAD method*), 334  
`get_parent()` (*MMVAD\_SHORT method*), 336  
`get_path()` (*vm\_map\_entry method*), 308  
`get_peb()` (*EPROCESS method*), 321  
`get_perms()` (*sysctl\_oid method*), 307  
`get_perms()` (*vm\_map\_entry method*), 308  
`get_physical_layer_name()` (*KernelPDBScanner method*), 56  
`get_pid()` (*SERVICE\_RECORD method*), 372

`get_pool_header_table()` (*PoolScanner class method*), 512

`get_pool_type()` (*POOL\_TRACKER\_BIG\_PAGES method*), 359

`get_private_memory()` (*MMVAD method*), 334

`get_private_memory()` (*MMVAD\_SHORT method*), 336

`get_process_memory_sections()` (*proc method*), 299

`get_process_memory_sections()` (*task\_struct method*), 274

`get_program_headers()` (*elf method*), 281

`get_protection()` (*MMVAD method*), 334

`get_protection()` (*MMVAD\_SHORT method*), 336

`get_protection()` (*vm\_area\_struct method*), 278

`get_protocol_as_string()` (*socket method*), 305

`get_pte()` (*CONTROL\_AREA method*), 316

`get_range_alias()` (*vm\_map\_entry method*), 308

`get_render_options()` (*CLIRenderer method*), 41

`get_render_options()` (*CSVRenderer method*), 41

`get_render_options()` (*JsonLinesRenderer method*), 41

`get_render_options()` (*JsonRenderer method*), 42

`get_render_options()` (*PrettyTextRenderer method*), 42

`get_render_options()` (*QuickTextRenderer method*), 42

`get_render_options()` (*Renderer method*), 130

`get_requirements()` (*AutomagicInterface class method*), 100

`get_requirements()` (*Banners class method*), 534

`get_requirements()` (*Bash class method*), 409, 434

`get_requirements()` (*BashIntermedSymbols class method*), 287

`get_requirements()` (*BigPools class method*), 478

`get_requirements()` (*BufferDataLayer class method*), 177

`get_requirements()` (*Check\_afinfo class method*), 411

`get_requirements()` (*Check\_creds class method*), 412

`get_requirements()` (*Check\_idt class method*), 414

`get_requirements()` (*Check\_modules class method*), 415

`get_requirements()` (*Check\_syscall class method*), 417, 435

`get_requirements()` (*Check\_sysctl class method*), 437

`get_requirements()` (*Check\_trap\_table class method*), 438

`get_requirements()` (*CmdLine class method*), 480

`get_requirements()` (*ComplexListRequirement class method*), 75

`get_requirements()` (*ConfigurableInterface class method*), 103

`get_requirements()` (*ConfigWriter class method*), 535

`get_requirements()` (*ConstructionMagic class method*), 47

`get_requirements()` (*DataLayerInterface class method*), 116

`get_requirements()` (*DllList class method*), 482

`get_requirements()` (*DriverIrp class method*), 483

`get_requirements()` (*DriverScan class method*), 485

`get_requirements()` (*DumpFiles class method*), 487

`get_requirements()` (*Elf64Layer class method*), 147

`get_requirements()` (*Elfs class method*), 418

`get_requirements()` (*Envvars class method*), 488

`get_requirements()` (*FileLayer class method*), 179

`get_requirements()` (*FileScan class method*), 490

`get_requirements()` (*FrameworkInfo class method*), 537

`get_requirements()` (*GetServiceSIDs class method*), 492

`get_requirements()` (*GetSIDs class method*), 493

`get_requirements()` (*Handles class method*), 495

`get_requirements()` (*HiveList class method*), 470

`get_requirements()` (*HiveScan class method*), 472

`get_requirements()` (*Ifconfig class method*), 440

`get_requirements()` (*Info class method*), 497

`get_requirements()` (*Intel class method*), 150

`get_requirements()` (*Intel32e class method*), 153

`get_requirements()` (*IntelPAE class method*), 155

`get_requirements()` (*IntermediateSymbolTable class method*), 382

`get_requirements()` (*IsfInfo class method*), 538

`get_requirements()` (*ISFormatTable class method*), 378

`get_requirements()` (*Kauth\_listeners class method*), 441

`get_requirements()` (*Kauth\_scopes class method*), 443

`get_requirements()` (*KernelPDBScanner class method*), 56

`get_requirements()` (*Kevents class method*), 444

`get_requirements()` (*Keyboard\_notifiers class method*), 420

`get_requirements()` (*LayerListRequirement class method*), 78

`get_requirements()` (*LayerStacker class method*), 59

`get_requirements()` (*LayerWriter class method*), 540

`get_requirements()` (*LimeLayer class method*),

- 166
- `get_requirements()` (*LinearlyMappedLayer* class method), 169
- `get_requirements()` (*LinuxBannerCache* class method), 48
- `get_requirements()` (*LinuxKernelIntermedSymbols* class method), 255
- `get_requirements()` (*LinuxSymbolFinder* class method), 50
- `get_requirements()` (*List\_Files* class method), 446
- `get_requirements()` (*Lsmmod* class method), 422, 448
- `get_requirements()` (*Lsof* class method), 423, 449
- `get_requirements()` (*MacBannerCache* class method), 52
- `get_requirements()` (*MacKernelIntermedSymbols* class method), 290
- `get_requirements()` (*MacSymbolFinder* class method), 54
- `get_requirements()` (*Malfind* class method), 425, 451, 499
- `get_requirements()` (*Maps* class method), 426, 456
- `get_requirements()` (*Memmap* class method), 501
- `get_requirements()` (*ModScan* class method), 503
- `get_requirements()` (*Modules* class method), 505
- `get_requirements()` (*Mount* class method), 452
- `get_requirements()` (*MutantScan* class method), 507
- `get_requirements()` (*NetScan* class method), 510
- `get_requirements()` (*Netstat* class method), 454
- `get_requirements()` (*NonLinearlySegmentedLayer* class method), 189
- `get_requirements()` (*PdbMSFStream* class method), 172
- `get_requirements()` (*PdbMultiStreamFormat* class method), 174
- `get_requirements()` (*PluginInterface* class method), 128
- `get_requirements()` (*PoolScanner* class method), 513
- `get_requirements()` (*PrintKey* class method), 474
- `get_requirements()` (*Privs* class method), 515
- `get_requirements()` (*Psaux* class method), 457
- `get_requirements()` (*PsList* class method), 428, 459, 517
- `get_requirements()` (*PsScan* class method), 519
- `get_requirements()` (*PsTree* class method), 430, 462, 521
- `get_requirements()` (*QemuSuspendLayer* class method), 182
- `get_requirements()` (*RegistryHive* class method), 185
- `get_requirements()` (*SegmentedLayer* class method), 191
- `get_requirements()` (*Socket\_filters* class method), 464
- `get_requirements()` (*SSDT* class method), 523
- `get_requirements()` (*Strings* class method), 524
- `get_requirements()` (*SymbolBannerCache* class method), 61
- `get_requirements()` (*SymbolFinder* class method), 63
- `get_requirements()` (*SymbolTableInterface* class method), 138
- `get_requirements()` (*SymlinkScan* class method), 526
- `get_requirements()` (*Timeliner* class method), 542
- `get_requirements()` (*Timers* class method), 465
- `get_requirements()` (*TranslationLayerInterface* class method), 121
- `get_requirements()` (*Trustedbsd* class method), 467
- `get_requirements()` (*try\_check* class method), 432
- `get_requirements()` (*UserAssist* class method), 476
- `get_requirements()` (*VadInfo* class method), 528
- `get_requirements()` (*VerInfo* class method), 530
- `get_requirements()` (*Version1Format* class method), 384
- `get_requirements()` (*Version2Format* class method), 387
- `get_requirements()` (*Version3Format* class method), 389
- `get_requirements()` (*Version4Format* class method), 392
- `get_requirements()` (*Version5Format* class method), 395
- `get_requirements()` (*Version6Format* class method), 397
- `get_requirements()` (*Version7Format* class method), 400
- `get_requirements()` (*Version8Format* class method), 402
- `get_requirements()` (*VFSEvents* class method), 468
- `get_requirements()` (*VirtMap* class method), 532
- `get_requirements()` (*VmwareLayer* class method), 193
- `get_requirements()` (*Volshell* class method), 33, 35, 37, 39
- `get_requirements()` (*WindowsCrash-Dump32Layer* class method), 142
- `get_requirements()` (*WindowsCrash-Dump64Layer* class method), 144
- `get_requirements()` (*WindowsIntel* class method), 157
- `get_requirements()` (*WindowsIntel32e* class

*method*), 159  
get\_requirements() (*WindowsIntelPAE class method*), 162  
get\_requirements() (*WindowsKernelIntermedSymbols class method*), 314  
get\_requirements() (*WindowsMixin class method*), 164  
get\_requirements() (*WinSwapLayers class method*), 67  
get\_requirements() (*WintelHelper class method*), 69  
get\_right\_child() (*MMVAD method*), 335  
get\_right\_child() (*MMVAD\_SHORT method*), 337  
get\_root\_dentry() (*fs\_struct method*), 260  
get\_root\_mnt() (*fs\_struct method*), 260  
get\_section\_headers() (*elf method*), 281  
get\_sections() (*IMAGE\_NT\_HEADERS method*), 351  
get\_sections() (*module method*), 266  
get\_session\_id() (*EPROCESS method*), 321  
get\_session\_layers() (*ModScan class method*), 503  
get\_session\_layers() (*Modules class method*), 505  
get\_sids() (*TOKEN method*), 342  
get\_size\_from\_index() (*PdbReader method*), 374  
get\_special\_path() (*vm\_map\_entry method*), 308  
get\_start() (*MMVAD method*), 335  
get\_start() (*MMVAD\_SHORT method*), 337  
get\_state() (*KTHREAD method*), 331  
get\_state() (*socket method*), 305  
get\_stream() (*PdbMultiStreamFormat method*), 174  
get\_string() (*UNICODE\_STRING method*), 344  
get\_subkeys() (*CM\_KEY\_NODE method*), 365  
get\_subsection() (*CONTROL\_AREA method*), 316  
get\_symbol() (*BaseSymbolTableInterface method*), 133  
get\_symbol() (*BashIntermedSymbols method*), 287  
get\_symbol() (*IntermediateSymbolTable method*), 382  
get\_symbol() (*ISFormatTable method*), 378  
get\_symbol() (*LinuxKernelIntermedSymbols method*), 255  
get\_symbol() (*MacKernelIntermedSymbols method*), 290  
get\_symbol() (*Module method*), 95  
get\_symbol() (*module method*), 267  
get\_symbol() (*ModuleInterface method*), 114  
get\_symbol() (*NativeTable method*), 405  
get\_symbol() (*NativeTableInterface method*), 134  
get\_symbol() (*SizedModule method*), 97  
get\_symbol() (*SymbolSpace method*), 250  
get\_symbol() (*SymbolSpaceInterface method*), 136  
get\_symbol() (*SymbolTableInterface method*), 138  
get\_symbol() (*Version1Format method*), 384  
get\_symbol() (*Version2Format method*), 387  
get\_symbol() (*Version3Format method*), 389  
get\_symbol() (*Version4Format method*), 392  
get\_symbol() (*Version5Format method*), 395  
get\_symbol() (*Version6Format method*), 397  
get\_symbol() (*Version7Format method*), 400  
get\_symbol() (*Version8Format method*), 402  
get\_symbol() (*WindowsKernelIntermedSymbols method*), 314  
get\_symbol\_table\_name() (*AggregateType method*), 196  
get\_symbol\_table\_name() (*Array method*), 198  
get\_symbol\_table\_name() (*BitField method*), 200  
get\_symbol\_table\_name() (*Boolean method*), 202  
get\_symbol\_table\_name() (*Bytes method*), 204  
get\_symbol\_table\_name() (*Char method*), 209  
get\_symbol\_table\_name() (*ClassType method*), 211  
get\_symbol\_table\_name() (*CM\_KEY\_BODY method*), 363  
get\_symbol\_table\_name() (*CM\_KEY\_NODE method*), 365  
get\_symbol\_table\_name() (*CM\_KEY\_VALUE method*), 366  
get\_symbol\_table\_name() (*CMHIVE method*), 361  
get\_symbol\_table\_name() (*CONTROL\_AREA method*), 316  
get\_symbol\_table\_name() (*dentry method*), 257  
get\_symbol\_table\_name() (*DEVICE\_OBJECT method*), 318  
get\_symbol\_table\_name() (*DRIVER\_OBJECT method*), 319  
get\_symbol\_table\_name() (*elf method*), 281  
get\_symbol\_table\_name() (*elf\_phdr method*), 283  
get\_symbol\_table\_name() (*elf\_sym method*), 284  
get\_symbol\_table\_name() (*Enumeration method*), 213  
get\_symbol\_table\_name() (*EPROCESS method*), 321  
get\_symbol\_table\_name() (*ETHREAD method*), 323  
get\_symbol\_table\_name() (*EX\_FAST\_REF method*), 325  
get\_symbol\_table\_name() (*ExecutiveObject method*), 353  
get\_symbol\_table\_name() (*FILE\_OBJECT method*), 326



`get_symbol_table_name()` (*fileglob method*), 293  
`get_symbol_table_name()` (*files\_struct method*), 259  
`get_symbol_table_name()` (*Float method*), 215  
`get_symbol_table_name()` (*fs\_struct method*), 260  
`get_symbol_table_name()` (*Function method*), 217  
`get_symbol_table_name()` (*GenericIntelProcess method*), 252  
`get_symbol_table_name()` (*hist\_entry method*), 279  
`get_symbol_table_name()` (*HMAP\_ENTRY method*), 368  
`get_symbol_table_name()` (*ifnet method*), 294  
`get_symbol_table_name()` (*IM-AGE\_DOS\_HEADER method*), 349  
`get_symbol_table_name()` (*IM-AGE\_NT\_HEADERS method*), 351  
`get_symbol_table_name()` (*inpcb method*), 295  
`get_symbol_table_name()` (*Integer method*), 219  
`get_symbol_table_name()` (*kauth\_scope method*), 297  
`get_symbol_table_name()` (*KDDEBUG-GER\_DATA64 method*), 347  
`get_symbol_table_name()` (*KMUTANT method*), 328  
`get_symbol_table_name()` (*kobject method*), 262  
`get_symbol_table_name()` (*KSYSTEM\_TIME method*), 330  
`get_symbol_table_name()` (*KTHREAD method*), 331  
`get_symbol_table_name()` (*LIST\_ENTRY method*), 333  
`get_symbol_table_name()` (*list\_head method*), 263  
`get_symbol_table_name()` (*mm\_struct method*), 265  
`get_symbol_table_name()` (*MMVAD method*), 335  
`get_symbol_table_name()` (*MMVAD\_SHORT method*), 337  
`get_symbol_table_name()` (*module method*), 267  
`get_symbol_table_name()` (*mount method*), 268  
`get_symbol_table_name()` (*OBJECT\_HEADER method*), 354  
`get_symbol_table_name()` (*OBJECT\_SYMBOLIC\_LINK method*), 338  
`get_symbol_table_name()` (*ObjectInterface method*), 124  
`get_symbol_table_name()` (*Pointer method*), 221  
`get_symbol_table_name()` (*POOL\_HEADER method*), 356  
`get_symbol_table_name()` (*POOL\_HEADER\_VISTA method*), 358  
`get_symbol_table_name()` (*POOL\_TRACKER\_BIG\_PAGES method*), 360  
`get_symbol_table_name()` (*PrimitiveObject method*), 223  
`get_symbol_table_name()` (*proc method*), 299  
`get_symbol_table_name()` (*qstr method*), 270  
`get_symbol_table_name()` (*queue\_entry method*), 300  
`get_symbol_table_name()` (*SERVICE\_HEADER method*), 370  
`get_symbol_table_name()` (*SERVICE\_RECORD method*), 372  
`get_symbol_table_name()` (*SHARED\_CACHE\_MAP method*), 340  
`get_symbol_table_name()` (*sockaddr method*), 302  
`get_symbol_table_name()` (*sockaddr\_dl method*), 303  
`get_symbol_table_name()` (*socket method*), 305  
`get_symbol_table_name()` (*String method*), 226  
`get_symbol_table_name()` (*struct\_file method*), 271  
`get_symbol_table_name()` (*StructType method*), 230  
`get_symbol_table_name()` (*super\_block method*), 273  
`get_symbol_table_name()` (*sysctl\_oid method*), 307  
`get_symbol_table_name()` (*task\_struct method*), 274  
`get_symbol_table_name()` (*TOKEN method*), 342  
`get_symbol_table_name()` (*UNICODE\_STRING method*), 344  
`get_symbol_table_name()` (*UnionType method*), 232  
`get_symbol_table_name()` (*VACB method*), 345  
`get_symbol_table_name()` (*vfsmount method*), 276  
`get_symbol_table_name()` (*vm\_area\_struct method*), 278  
`get_symbol_table_name()` (*vm\_map\_entry method*), 308  
`get_symbol_table_name()` (*vm\_map\_object method*), 310  
`get_symbol_table_name()` (*vnode method*), 311  
`get_symbol_table_name()` (*Void method*), 233  
`get_symbol_type()` (*BaseSymbolTableInterface method*), 133  
`get_symbol_type()` (*BashIntermedSymbols method*), 287  
`get_symbol_type()` (*IntermediateSymbolTable method*), 382

`get_symbol_type()` (*ISFormatTable method*), 378  
`get_symbol_type()` (*LinuxKernelIntermedSymbols method*), 255  
`get_symbol_type()` (*MacKernelIntermedSymbols method*), 290  
`get_symbol_type()` (*NativeTable method*), 405  
`get_symbol_type()` (*NativeTableInterface method*), 135  
`get_symbol_type()` (*SymbolTableInterface method*), 138  
`get_symbol_type()` (*Version1Format method*), 384  
`get_symbol_type()` (*Version2Format method*), 387  
`get_symbol_type()` (*Version3Format method*), 389  
`get_symbol_type()` (*Version4Format method*), 392  
`get_symbol_type()` (*Version5Format method*), 395  
`get_symbol_type()` (*Version6Format method*), 397  
`get_symbol_type()` (*Version7Format method*), 400  
`get_symbol_type()` (*Version8Format method*), 403  
`get_symbol_type()` (*WindowsKernelIntermedSymbols method*), 314  
`get_symbols()` (*elf method*), 281  
`get_symbols()` (*module method*), 267  
`get_symbols_by_absolute_location()` (*SizedModule method*), 97  
`get_symbols_by_location()` (*BaseSymbolTableInterface method*), 133  
`get_symbols_by_location()` (*BashIntermedSymbols method*), 287  
`get_symbols_by_location()` (*IntermediateSymbolTable method*), 382  
`get_symbols_by_location()` (*ISFormatTable method*), 379  
`get_symbols_by_location()` (*LinuxKernelIntermedSymbols method*), 255  
`get_symbols_by_location()` (*MacKernelIntermedSymbols method*), 290  
`get_symbols_by_location()` (*NativeTable method*), 405  
`get_symbols_by_location()` (*NativeTableInterface method*), 135  
`get_symbols_by_location()` (*SymbolSpace method*), 251  
`get_symbols_by_location()` (*SymbolSpaceInterface method*), 136  
`get_symbols_by_location()` (*SymbolTableInterface method*), 138  
`get_symbols_by_location()` (*Version1Format method*), 384  
`get_symbols_by_location()` (*Version2Format method*), 387  
`get_symbols_by_location()` (*Version3Format method*), 390  
`get_symbols_by_location()` (*Version4Format method*), 392  
`get_symbols_by_location()` (*Version5Format method*), 395  
`get_symbols_by_location()` (*Version6Format method*), 397  
`get_symbols_by_location()` (*Version7Format method*), 400  
`get_symbols_by_location()` (*Version8Format method*), 403  
`get_symbols_by_location()` (*WindowsKernelIntermedSymbols method*), 314  
`get_tag()` (*MMVAD method*), 335  
`get_tag()` (*MMVAD\_SHORT method*), 337  
`get_task()` (*proc method*), 299  
`get_tcp_state()` (*inpcb method*), 296  
`get_symbols_by_location()` (*Version5Format method*), 395  
`get_symbols_by_location()` (*Version6Format method*), 397  
`get_symbols_by_location()` (*Version7Format method*), 400  
`get_symbols_by_location()` (*Version8Format method*), 403  
`get_symbols_by_location()` (*WindowsKernelIntermedSymbols method*), 314  
`get_symbols_by_type()` (*BaseSymbolTableInterface method*), 133  
`get_symbols_by_type()` (*BashIntermedSymbols method*), 287  
`get_symbols_by_type()` (*IntermediateSymbolTable method*), 382  
`get_symbols_by_type()` (*ISFormatTable method*), 379  
`get_symbols_by_type()` (*LinuxKernelIntermedSymbols method*), 255  
`get_symbols_by_type()` (*MacKernelIntermedSymbols method*), 290  
`get_symbols_by_type()` (*NativeTable method*), 405  
`get_symbols_by_type()` (*NativeTableInterface method*), 135  
`get_symbols_by_type()` (*SymbolSpace method*), 251  
`get_symbols_by_type()` (*SymbolSpaceInterface method*), 136  
`get_symbols_by_type()` (*SymbolTableInterface method*), 138  
`get_symbols_by_type()` (*Version1Format method*), 384  
`get_symbols_by_type()` (*Version2Format method*), 387  
`get_symbols_by_type()` (*Version3Format method*), 390  
`get_symbols_by_type()` (*Version4Format method*), 392  
`get_symbols_by_type()` (*Version5Format method*), 395  
`get_symbols_by_type()` (*Version6Format method*), 397  
`get_symbols_by_type()` (*Version7Format method*), 400  
`get_symbols_by_type()` (*Version8Format method*), 403  
`get_symbols_by_type()` (*WindowsKernelIntermedSymbols method*), 314

- `get_time()` (*KSYSTEM\_TIME* method), 330
  - `get_time_as_integer()` (*hist\_entry* method), 280
  - `get_time_object()` (*hist\_entry* method), 280
  - `get_type()` (*BaseSymbolTableInterface* method), 133
  - `get_type()` (*BashIntermedSymbols* method), 287
  - `get_type()` (*IntermediateSymbolTable* method), 382
  - `get_type()` (*ISFormatTable* method), 379
  - `get_type()` (*LinuxKernelIntermedSymbols* method), 255
  - `get_type()` (*MacKernelIntermedSymbols* method), 290
  - `get_type()` (*Module* method), 95
  - `get_type()` (*ModuleInterface* method), 114
  - `get_type()` (*NativeTable* method), 405
  - `get_type()` (*NativeTableInterface* method), 135
  - `get_type()` (*SERVICE\_RECORD* method), 372
  - `get_type()` (*SizedModule* method), 97
  - `get_type()` (*SymbolSpace* method), 251
  - `get_type()` (*SymbolSpaceInterface* method), 137
  - `get_type()` (*SymbolTableInterface* method), 138
  - `get_type()` (*Version1Format* method), 385
  - `get_type()` (*Version2Format* method), 387
  - `get_type()` (*Version3Format* method), 390
  - `get_type()` (*Version4Format* method), 392
  - `get_type()` (*Version5Format* method), 395
  - `get_type()` (*Version6Format* method), 398
  - `get_type()` (*Version7Format* method), 400
  - `get_type()` (*Version8Format* method), 403
  - `get_type()` (*WindowsKernelIntermedSymbols* method), 314
  - `get_type_class()` (*BaseSymbolTableInterface* method), 133
  - `get_type_class()` (*BashIntermedSymbols* method), 287
  - `get_type_class()` (*IntermediateSymbolTable* method), 382
  - `get_type_class()` (*ISFormatTable* method), 379
  - `get_type_class()` (*LinuxKernelIntermedSymbols* method), 255
  - `get_type_class()` (*MacKernelIntermedSymbols* method), 290
  - `get_type_class()` (*NativeTable* method), 405
  - `get_type_class()` (*NativeTableInterface* method), 135
  - `get_type_class()` (*SymbolTableInterface* method), 138
  - `get_type_class()` (*Version1Format* method), 385
  - `get_type_class()` (*Version2Format* method), 387
  - `get_type_class()` (*Version3Format* method), 390
  - `get_type_class()` (*Version4Format* method), 392
  - `get_type_class()` (*Version5Format* method), 395
  - `get_type_class()` (*Version6Format* method), 398
  - `get_type_class()` (*Version7Format* method), 400
  - `get_type_class()` (*Version8Format* method), 403
  - `get_type_class()` (*WindowsKernelIntermedSymbols* method), 314
  - `get_vad_root()` (*EPROCESS* method), 321
  - `get_vaddr()` (*elf\_phdr* method), 283
  - `get_values()` (*CM\_KEY\_NODE* method), 365
  - `get_version_information()` (*VerInfo* class method), 530
  - `get_version_structure()` (*Info* class method), 497
  - `get_vfsmnt()` (*struct\_file* method), 271
  - `get_vnode()` (*vm\_map\_entry* method), 308
  - `get_volatile()` (*CM\_KEY\_NODE* method), 365
  - `get_wait_reason()` (*KTHREAD* method), 331
  - `get_wow_64_process()` (*EPROCESS* method), 322
  - `getbuffer()` (*NullFileHandler* method), 30
  - `GetServiceSIDs` (class in *volatility.plugins.windows.getservicesids*), 491
  - `GetSIDs` (class in *volatility.plugins.windows.getsids*), 493
  - `getter()` (*classproperty* method), 27
  - `getvalue()` (*NullFileHandler* method), 30
  - `group_structure` (*VmwareLayer* attribute), 193
- ## H
- `handler_order` (*JarHandler* attribute), 187
  - `Handles` (class in *volatility.plugins.windows.handles*), 494
  - `handles()` (*Handles* method), 495
  - `has_enumeration()` (*Module* method), 95
  - `has_enumeration()` (*ModuleInterface* method), 114
  - `has_enumeration()` (*SizedModule* method), 97
  - `has_enumeration()` (*SymbolSpace* method), 251
  - `has_enumeration()` (*SymbolSpaceInterface* method), 137
  - `has_member()` (*AggregateType* method), 196
  - `has_member()` (*AggregateType.VolTemplateProxy* class method), 196
  - `has_member()` (*Array* method), 198
  - `has_member()` (*Array.VolTemplateProxy* class method), 197
  - `has_member()` (*BitField* method), 200
  - `has_member()` (*BitField.VolTemplateProxy* class method), 199
  - `has_member()` (*Boolean* method), 202
  - `has_member()` (*Boolean.VolTemplateProxy* class method), 201
  - `has_member()` (*Bytes* method), 204
  - `has_member()` (*Bytes.VolTemplateProxy* class method), 203

`has_member()` (*Char* method), 209  
`has_member()` (*Char.VolTemplateProxy* class method), 208  
`has_member()` (*ClassType* method), 211  
`has_member()` (*ClassType.VolTemplateProxy* class method), 210  
`has_member()` (*CM\_KEY\_BODY* method), 363  
`has_member()` (*CM\_KEY\_BODY.VolTemplateProxy* class method), 363  
`has_member()` (*CM\_KEY\_NODE* method), 365  
`has_member()` (*CM\_KEY\_NODE.VolTemplateProxy* class method), 364  
`has_member()` (*CM\_KEY\_VALUE* method), 367  
`has_member()` (*CM\_KEY\_VALUE.VolTemplateProxy* class method), 366  
`has_member()` (*CMHIVE* method), 362  
`has_member()` (*CMHIVE.VolTemplateProxy* class method), 361  
`has_member()` (*CONTROL\_AREA* method), 316  
`has_member()` (*CONTROL\_AREA.VolTemplateProxy* class method), 315  
`has_member()` (*dentry* method), 257  
`has_member()` (*dentry.VolTemplateProxy* class method), 257  
`has_member()` (*DEVICE\_OBJECT* method), 318  
`has_member()` (*DEVICE\_OBJECT.VolTemplateProxy* class method), 317  
`has_member()` (*DRIVER\_OBJECT* method), 320  
`has_member()` (*DRIVER\_OBJECT.VolTemplateProxy* class method), 319  
`has_member()` (*elf* method), 281  
`has_member()` (*elf.VolTemplateProxy* class method), 281  
`has_member()` (*elf\_phdr* method), 283  
`has_member()` (*elf\_phdr.VolTemplateProxy* class method), 282  
`has_member()` (*elf\_sym* method), 284  
`has_member()` (*elf\_sym.VolTemplateProxy* class method), 284  
`has_member()` (*Enumeration* method), 213  
`has_member()` (*Enumeration.VolTemplateProxy* class method), 212  
`has_member()` (*EPROCESS* method), 322  
`has_member()` (*EPROCESS.VolTemplateProxy* class method), 321  
`has_member()` (*ETHREAD* method), 323  
`has_member()` (*ETHREAD.VolTemplateProxy* class method), 323  
`has_member()` (*EX\_FAST\_REF* method), 325  
`has_member()` (*EX\_FAST\_REF.VolTemplateProxy* class method), 324  
`has_member()` (*ExecutiveObject* method), 353  
`has_member()` (*ExecutiveObject.VolTemplateProxy* class method), 352  
`has_member()` (*FILE\_OBJECT* method), 327  
`has_member()` (*FILE\_OBJECT.VolTemplateProxy* class method), 326  
`has_member()` (*fileglob* method), 293  
`has_member()` (*fileglob.VolTemplateProxy* class method), 292  
`has_member()` (*files\_struct* method), 259  
`has_member()` (*files\_struct.VolTemplateProxy* class method), 258  
`has_member()` (*Float* method), 215  
`has_member()` (*Float.VolTemplateProxy* class method), 214  
`has_member()` (*fs\_struct* method), 261  
`has_member()` (*fs\_struct.VolTemplateProxy* class method), 260  
`has_member()` (*Function* method), 217  
`has_member()` (*Function.VolTemplateProxy* class method), 217  
`has_member()` (*GenericIntelProcess* method), 252  
`has_member()` (*GenericIntelProcess.VolTemplateProxy* class method), 252  
`has_member()` (*hist\_entry* method), 280  
`has_member()` (*hist\_entry.VolTemplateProxy* class method), 279  
`has_member()` (*HMAP\_ENTRY* method), 368  
`has_member()` (*HMAP\_ENTRY.VolTemplateProxy* class method), 367  
`has_member()` (*ifnet* method), 294  
`has_member()` (*ifnet.VolTemplateProxy* class method), 294  
`has_member()` (*IMAGE\_DOS\_HEADER* method), 349  
`has_member()` (*IMAGE\_DOS\_HEADER.VolTemplateProxy* class method), 348  
`has_member()` (*IMAGE\_NT\_HEADERS* method), 351  
`has_member()` (*IMAGE\_NT\_HEADERS.VolTemplateProxy* class method), 350  
`has_member()` (*inpcb* method), 296  
`has_member()` (*inpcb.VolTemplateProxy* class method), 295  
`has_member()` (*Integer* method), 219  
`has_member()` (*Integer.VolTemplateProxy* class method), 218  
`has_member()` (*kauth\_scope* method), 297  
`has_member()` (*kauth\_scope.VolTemplateProxy* class method), 297  
`has_member()` (*KDDEBUGGER\_DATA64* method), 347  
`has_member()` (*KDDEBUGGER\_DATA64.VolTemplateProxy* class method), 346  
`has_member()` (*KMUTANT* method), 328  
`has_member()` (*KMUTANT.VolTemplateProxy* class method), 328



[has\\_member \(\) \(kobject method\), 262](#)  
[has\\_member \(\) \(kobject.VolTemplateProxy class method\), 261](#)  
[has\\_member \(\) \(KSYSTEM\\_TIME method\), 330](#)  
[has\\_member \(\) \(KSYSTEM\\_TIME.VolTemplateProxy class method\), 329](#)  
[has\\_member \(\) \(KTHREAD method\), 331](#)  
[has\\_member \(\) \(KTHREAD.VolTemplateProxy class method\), 331](#)  
[has\\_member \(\) \(LIST\\_ENTRY method\), 333](#)  
[has\\_member \(\) \(LIST\\_ENTRY.VolTemplateProxy class method\), 332](#)  
[has\\_member \(\) \(list\\_head method\), 263](#)  
[has\\_member \(\) \(list\\_head.VolTemplateProxy class method\), 263](#)  
[has\\_member \(\) \(mm\\_struct method\), 265](#)  
[has\\_member \(\) \(mm\\_struct.VolTemplateProxy class method\), 264](#)  
[has\\_member \(\) \(MMVAD method\), 335](#)  
[has\\_member \(\) \(MMVAD.VolTemplateProxy class method\), 334](#)  
[has\\_member \(\) \(MMVAD\\_SHORT method\), 337](#)  
[has\\_member \(\) \(MMVAD\\_SHORT.VolTemplateProxy class method\), 336](#)  
[has\\_member \(\) \(module method\), 267](#)  
[has\\_member \(\) \(module.VolTemplateProxy class method\), 266](#)  
[has\\_member \(\) \(mount method\), 268](#)  
[has\\_member \(\) \(mount.VolTemplateProxy class method\), 268](#)  
[has\\_member \(\) \(OBJECT\\_HEADER method\), 354](#)  
[has\\_member \(\) \(OBJECT\\_HEADER.VolTemplateProxy class method\), 354](#)  
[has\\_member \(\) \(OBJECT\\_SYMBOLIC\\_LINK method\), 338](#)  
[has\\_member \(\) \(OBJECT\\_SYMBOLIC\\_LINK.VolTemplateProxy class method\), 338](#)  
[has\\_member \(\) \(ObjectInterface method\), 124](#)  
[has\\_member \(\) \(ObjectInterface.VolTemplateProxy class method\), 123](#)  
[has\\_member \(\) \(ObjectTemplate method\), 235](#)  
[has\\_member \(\) \(Pointer method\), 222](#)  
[has\\_member \(\) \(Pointer.VolTemplateProxy class method\), 220](#)  
[has\\_member \(\) \(POOL\\_HEADER method\), 356](#)  
[has\\_member \(\) \(POOL\\_HEADER.VolTemplateProxy class method\), 355](#)  
[has\\_member \(\) \(POOL\\_HEADER\\_VISTA method\), 358](#)  
[has\\_member \(\) \(POOL\\_HEADER\\_VISTA.VolTemplateProxy class method\), 357](#)  
[has\\_member \(\) \(POOL\\_TRACKER\\_BIG\\_PAGES method\), 360](#)  
[has\\_member \(\) \(POOL\\_TRACKER\\_BIG\\_PAGES.VolTemplateProxy class method\), 359](#)  
[has\\_member \(\) \(PrimitiveObject method\), 223](#)  
[has\\_member \(\) \(PrimitiveObject.VolTemplateProxy class method\), 223](#)  
[has\\_member \(\) \(proc method\), 299](#)  
[has\\_member \(\) \(proc.VolTemplateProxy class method\), 298](#)  
[has\\_member \(\) \(qstr method\), 270](#)  
[has\\_member \(\) \(qstr.VolTemplateProxy class method\), 269](#)  
[has\\_member \(\) \(queue\\_entry method\), 300](#)  
[has\\_member \(\) \(queue\\_entry.VolTemplateProxy class method\), 300](#)  
[has\\_member \(\) \(ReferenceTemplate method\), 235](#)  
[has\\_member \(\) \(SERVICE\\_HEADER method\), 370](#)  
[has\\_member \(\) \(SERVICE\\_HEADER.VolTemplateProxy class method\), 370](#)  
[has\\_member \(\) \(SERVICE\\_RECORD method\), 372](#)  
[has\\_member \(\) \(SERVICE\\_RECORD.VolTemplateProxy class method\), 371](#)  
[has\\_member \(\) \(SHARED\\_CACHE\\_MAP method\), 340](#)  
[has\\_member \(\) \(SHARED\\_CACHE\\_MAP.VolTemplateProxy class method\), 340](#)  
[has\\_member \(\) \(sockaddr method\), 302](#)  
[has\\_member \(\) \(sockaddr.VolTemplateProxy class method\), 302](#)  
[has\\_member \(\) \(sockaddr\\_dl method\), 304](#)  
[has\\_member \(\) \(sockaddr\\_dl.VolTemplateProxy class method\), 303](#)  
[has\\_member \(\) \(socket method\), 305](#)  
[has\\_member \(\) \(socket.VolTemplateProxy class method\), 304](#)  
[has\\_member \(\) \(String method\), 226](#)  
[has\\_member \(\) \(String.VolTemplateProxy class method\), 224](#)  
[has\\_member \(\) \(struct\\_file method\), 271](#)  
[has\\_member \(\) \(struct\\_file.VolTemplateProxy class method\), 271](#)  
[has\\_member \(\) \(StructType method\), 230](#)  
[has\\_member \(\) \(StructType.VolTemplateProxy class method\), 230](#)  
[has\\_member \(\) \(super\\_block method\), 273](#)  
[has\\_member \(\) \(super\\_block.VolTemplateProxy class method\), 272](#)  
[has\\_member \(\) \(SymbolSpace.UnresolvedTemplate method\), 250](#)  
[has\\_member \(\) \(sysctl\\_oid method\), 307](#)  
[has\\_member \(\) \(sysctl\\_oid.VolTemplateProxy class method\), 306](#)

`has_member()` (*task\_struct* method), 275  
`has_member()` (*task\_struct.VolTemplateProxy* class method), 274  
`has_member()` (*Template* method), 125  
`has_member()` (*TOKEN* method), 342  
`has_member()` (*TOKEN.VolTemplateProxy* class method), 342  
`has_member()` (*UNICODE\_STRING* method), 344  
`has_member()` (*UNICODE\_STRING.VolTemplateProxy* class method), 343  
`has_member()` (*UnionType* method), 232  
`has_member()` (*UnionType.VolTemplateProxy* class method), 231  
`has_member()` (*VACB* method), 345  
`has_member()` (*VACB.VolTemplateProxy* class method), 345  
`has_member()` (*vfsmount* method), 276  
`has_member()` (*vfsmount.VolTemplateProxy* class method), 275  
`has_member()` (*vm\_area\_struct* method), 278  
`has_member()` (*vm\_area\_struct.VolTemplateProxy* class method), 277  
`has_member()` (*vm\_map\_entry* method), 308  
`has_member()` (*vm\_map\_entry.VolTemplateProxy* class method), 308  
`has_member()` (*vm\_map\_object* method), 310  
`has_member()` (*vm\_map\_object.VolTemplateProxy* class method), 309  
`has_member()` (*vnode* method), 311  
`has_member()` (*vnode.VolTemplateProxy* class method), 311  
`has_member()` (*Void* method), 233  
`has_member()` (*Void.VolTemplateProxy* class method), 233  
`has_symbol()` (*Module* method), 95  
`has_symbol()` (*ModuleInterface* method), 114  
`has_symbol()` (*SizedModule* method), 97  
`has_symbol()` (*SymbolSpace* method), 251  
`has_symbol()` (*SymbolSpaceInterface* method), 137  
`has_type()` (*Module* method), 95  
`has_type()` (*ModuleInterface* method), 114  
`has_type()` (*SizedModule* method), 97  
`has_type()` (*SymbolSpace* method), 251  
`has_type()` (*SymbolSpaceInterface* method), 137  
`has_valid_member()` (*AggregateType* method), 196  
`has_valid_member()` (*Array* method), 198  
`has_valid_member()` (*BitField* method), 200  
`has_valid_member()` (*Boolean* method), 202  
`has_valid_member()` (*Bytes* method), 205  
`has_valid_member()` (*Char* method), 209  
`has_valid_member()` (*ClassType* method), 211  
`has_valid_member()` (*CM\_KEY\_BODY* method), 363  
`has_valid_member()` (*CM\_KEY\_NODE* method), 365  
`has_valid_member()` (*CM\_KEY\_VALUE* method), 367  
`has_valid_member()` (*CMHIVE* method), 362  
`has_valid_member()` (*CONTROL\_AREA* method), 316  
`has_valid_member()` (*dentry* method), 258  
`has_valid_member()` (*DEVICE\_OBJECT* method), 318  
`has_valid_member()` (*DRIVER\_OBJECT* method), 320  
`has_valid_member()` (*elf* method), 281  
`has_valid_member()` (*elf\_phdr* method), 283  
`has_valid_member()` (*elf\_sym* method), 284  
`has_valid_member()` (*Enumeration* method), 213  
`has_valid_member()` (*EPROCESS* method), 322  
`has_valid_member()` (*ETHREAD* method), 323  
`has_valid_member()` (*EX\_FAST\_REF* method), 325  
`has_valid_member()` (*ExecutiveObject* method), 353  
`has_valid_member()` (*FILE\_OBJECT* method), 327  
`has_valid_member()` (*fileglob* method), 293  
`has_valid_member()` (*files\_struct* method), 259  
`has_valid_member()` (*Float* method), 216  
`has_valid_member()` (*fs\_struct* method), 261  
`has_valid_member()` (*Function* method), 217  
`has_valid_member()` (*GenericIntelProcess* method), 253  
`has_valid_member()` (*hist\_entry* method), 280  
`has_valid_member()` (*HMAP\_ENTRY* method), 368  
`has_valid_member()` (*ifnet* method), 294  
`has_valid_member()` (*IMAGE\_DOS\_HEADER* method), 349  
`has_valid_member()` (*IMAGE\_NT\_HEADERS* method), 351  
`has_valid_member()` (*inpcb* method), 296  
`has_valid_member()` (*Integer* method), 219  
`has_valid_member()` (*kauth\_scope* method), 297  
`has_valid_member()` (*KDDEBUGGER\_DATA64* method), 347  
`has_valid_member()` (*KMUTANT* method), 328  
`has_valid_member()` (*kobject* method), 262  
`has_valid_member()` (*KSYSTEM\_TIME* method), 330  
`has_valid_member()` (*KTHREAD* method), 331  
`has_valid_member()` (*LIST\_ENTRY* method), 333  
`has_valid_member()` (*list\_head* method), 264  
`has_valid_member()` (*mm\_struct* method), 265  
`has_valid_member()` (*MMVAD* method), 335  
`has_valid_member()` (*MMVAD\_SHORT* method), 337

[has\\_valid\\_member\(\) \(module method\), 267](#)  
[has\\_valid\\_member\(\) \(mount method\), 268](#)  
[has\\_valid\\_member\(\) \(OBJECT\\_HEADER method\), 354](#)  
[has\\_valid\\_member\(\) \(OBJECT\\_SYMBOLIC\\_LINK method\), 339](#)  
[has\\_valid\\_member\(\) \(ObjectInterface method\), 124](#)  
[has\\_valid\\_member\(\) \(Pointer method\), 222](#)  
[has\\_valid\\_member\(\) \(POOL\\_HEADER method\), 356](#)  
[has\\_valid\\_member\(\) \(POOL\\_HEADER\\_VISTA method\), 358](#)  
[has\\_valid\\_member\(\) \(POOL\\_TRACKER\\_BIG\\_PAGES method\), 360](#)  
[has\\_valid\\_member\(\) \(PrimitiveObject method\), 224](#)  
[has\\_valid\\_member\(\) \(proc method\), 299](#)  
[has\\_valid\\_member\(\) \(qstr method\), 270](#)  
[has\\_valid\\_member\(\) \(queue\\_entry method\), 300](#)  
[has\\_valid\\_member\(\) \(SERVICE\\_HEADER method\), 371](#)  
[has\\_valid\\_member\(\) \(SERVICE\\_RECORD method\), 372](#)  
[has\\_valid\\_member\(\) \(SHARED\\_CACHE\\_MAP method\), 340](#)  
[has\\_valid\\_member\(\) \(sockaddr method\), 302](#)  
[has\\_valid\\_member\(\) \(sockaddr\\_dl method\), 304](#)  
[has\\_valid\\_member\(\) \(socket method\), 305](#)  
[has\\_valid\\_member\(\) \(String method\), 226](#)  
[has\\_valid\\_member\(\) \(struct\\_file method\), 271](#)  
[has\\_valid\\_member\(\) \(StructType method\), 230](#)  
[has\\_valid\\_member\(\) \(super\\_block method\), 273](#)  
[has\\_valid\\_member\(\) \(sysctl\\_oid method\), 307](#)  
[has\\_valid\\_member\(\) \(task\\_struct method\), 275](#)  
[has\\_valid\\_member\(\) \(TOKEN method\), 342](#)  
[has\\_valid\\_member\(\) \(UNICODE\\_STRING method\), 344](#)  
[has\\_valid\\_member\(\) \(UnionType method\), 232](#)  
[has\\_valid\\_member\(\) \(VACB method\), 345](#)  
[has\\_valid\\_member\(\) \(vfsmount method\), 276](#)  
[has\\_valid\\_member\(\) \(vm\\_area\\_struct method\), 278](#)  
[has\\_valid\\_member\(\) \(vm\\_map\\_entry method\), 309](#)  
[has\\_valid\\_member\(\) \(vm\\_map\\_object method\), 310](#)  
[has\\_valid\\_member\(\) \(vnode method\), 311](#)  
[has\\_valid\\_member\(\) \(Void method\), 233](#)  
[has\\_valid\\_members\(\) \(AggregateType method\), 196](#)  
[has\\_valid\\_members\(\) \(Array method\), 198](#)  
[has\\_valid\\_members\(\) \(BitField method\), 200](#)  
[has\\_valid\\_members\(\) \(Boolean method\), 202](#)  
[has\\_valid\\_members\(\) \(Bytes method\), 205](#)  
[has\\_valid\\_members\(\) \(Char method\), 209](#)  
[has\\_valid\\_members\(\) \(ClassType method\), 211](#)  
[has\\_valid\\_members\(\) \(CM\\_KEY\\_BODY method\), 363](#)  
[has\\_valid\\_members\(\) \(CM\\_KEY\\_NODE method\), 365](#)  
[has\\_valid\\_members\(\) \(CM\\_KEY\\_VALUE method\), 367](#)  
[has\\_valid\\_members\(\) \(CMHIVE method\), 362](#)  
[has\\_valid\\_members\(\) \(CONTROL\\_AREA method\), 316](#)  
[has\\_valid\\_members\(\) \(dentry method\), 258](#)  
[has\\_valid\\_members\(\) \(DEVICE\\_OBJECT method\), 318](#)  
[has\\_valid\\_members\(\) \(DRIVER\\_OBJECT method\), 320](#)  
[has\\_valid\\_members\(\) \(elf method\), 281](#)  
[has\\_valid\\_members\(\) \(elf\\_phdr method\), 283](#)  
[has\\_valid\\_members\(\) \(elf\\_sym method\), 285](#)  
[has\\_valid\\_members\(\) \(Enumeration method\), 213](#)  
[has\\_valid\\_members\(\) \(EPROCESS method\), 322](#)  
[has\\_valid\\_members\(\) \(ETHREAD method\), 324](#)  
[has\\_valid\\_members\(\) \(EX\\_FAST\\_REF method\), 325](#)  
[has\\_valid\\_members\(\) \(ExecutiveObject method\), 353](#)  
[has\\_valid\\_members\(\) \(FILE\\_OBJECT method\), 327](#)  
[has\\_valid\\_members\(\) \(fileglob method\), 293](#)  
[has\\_valid\\_members\(\) \(files\\_struct method\), 259](#)  
[has\\_valid\\_members\(\) \(Float method\), 216](#)  
[has\\_valid\\_members\(\) \(fs\\_struct method\), 261](#)  
[has\\_valid\\_members\(\) \(Function method\), 218](#)  
[has\\_valid\\_members\(\) \(GenericIntelProcess method\), 253](#)  
[has\\_valid\\_members\(\) \(hist\\_entry method\), 280](#)  
[has\\_valid\\_members\(\) \(HMAP\\_ENTRY method\), 368](#)  
[has\\_valid\\_members\(\) \(ifnet method\), 294](#)  
[has\\_valid\\_members\(\) \(IMAGE\\_DOS\\_HEADER method\), 349](#)  
[has\\_valid\\_members\(\) \(IMAGE\\_NT\\_HEADERS method\), 351](#)  
[has\\_valid\\_members\(\) \(inpcb method\), 296](#)  
[has\\_valid\\_members\(\) \(Integer method\), 219](#)  
[has\\_valid\\_members\(\) \(kauth\\_scope method\), 297](#)  
[has\\_valid\\_members\(\) \(KDDEBUGGER\\_DATA64 method\), 347](#)  
[has\\_valid\\_members\(\) \(KMUTANT method\), 328](#)  
[has\\_valid\\_members\(\) \(kobject method\), 262](#)  
[has\\_valid\\_members\(\) \(KSYSTEM\\_TIME method\), 330](#)  
[has\\_valid\\_members\(\) \(KTHREAD method\), 332](#)

`has_valid_members()` (*LIST\_ENTRY* method), 333  
`has_valid_members()` (*list\_head* method), 264  
`has_valid_members()` (*mm\_struct* method), 265  
`has_valid_members()` (*MMVAD* method), 335  
`has_valid_members()` (*MMVAD\_SHORT* method), 337  
`has_valid_members()` (*module* method), 267  
`has_valid_members()` (*mount* method), 269  
`has_valid_members()` (*OBJECT\_HEADER* method), 355  
`has_valid_members()` (*OBJECT\_SYMBOLIC\_LINK* method), 339  
`has_valid_members()` (*ObjectInterface* method), 124  
`has_valid_members()` (*Pointer* method), 222  
`has_valid_members()` (*POOL\_HEADER* method), 356  
`has_valid_members()` (*POOL\_HEADER\_VISTA* method), 358  
`has_valid_members()` (*POOL\_TRACKER\_BIG\_PAGES* method), 360  
`has_valid_members()` (*PrimitiveObject* method), 224  
`has_valid_members()` (*proc* method), 299  
`has_valid_members()` (*qstr* method), 270  
`has_valid_members()` (*queue\_entry* method), 300  
`has_valid_members()` (*SERVICE\_HEADER* method), 371  
`has_valid_members()` (*SERVICE\_RECORD* method), 373  
`has_valid_members()` (*SHARED\_CACHE\_MAP* method), 340  
`has_valid_members()` (*sockaddr* method), 302  
`has_valid_members()` (*sockaddr\_dl* method), 304  
`has_valid_members()` (*socket* method), 305  
`has_valid_members()` (*String* method), 226  
`has_valid_members()` (*struct\_file* method), 272  
`has_valid_members()` (*StructType* method), 231  
`has_valid_members()` (*super\_block* method), 273  
`has_valid_members()` (*sysctl\_oid* method), 307  
`has_valid_members()` (*task\_struct* method), 275  
`has_valid_members()` (*TOKEN* method), 342  
`has_valid_members()` (*UNICODE\_STRING* method), 344  
`has_valid_members()` (*UnionType* method), 232  
`has_valid_members()` (*VACB* method), 346  
`has_valid_members()` (*vfsmount* method), 276  
`has_valid_members()` (*vm\_area\_struct* method), 278  
`has_valid_members()` (*vm\_map\_entry* method), 309  
`has_valid_members()` (*vm\_map\_object* method), 310  
`has_valid_members()` (*vnnode* method), 312  
`has_valid_members()` (*Void* method), 234  
`hash()` (*SizedModule* property), 97  
`HASH_PTE_SIZE_64` (*QemuSuspendLayer* attribute), 181  
`header_structure` (*VmwareLayer* attribute), 193  
`headerpages` (*WindowsCrashDump32Layer* attribute), 142  
`headerpages` (*WindowsCrashDump64Layer* attribute), 145  
`help()` (*Volshell* method), 33, 35, 37, 39  
`HelpfulArgParser` (class in *volatility.cli.volargparse*), 43  
`HelpfulSubparserAction` (class in *volatility.cli.volargparse*), 44  
`Hex` (class in *volatility.framework.renderers.format\_hints*), 242  
`hex()` (*Bytes* method), 205  
`hex()` (*Float* method), 216  
`hex()` (*HexBytes* method), 243  
`hex()` (*MultiTypeData* method), 246  
`hex_bytes_as_text()` (in module *volatility.cli.text\_renderer*), 43  
`HexBytes` (class in *volatility.framework.renderers.format\_hints*), 242  
`hide_from_subclasses()` (in module *volatility.framework*), 44  
`HierarchicalDict` (class in *volatility.framework.interfaces.configuration*), 107  
`hist_entry` (class in *volatility.framework.symbols.linux.extensions.bash*), 279  
`hist_entry.VolTemplateProxy` (class in *volatility.framework.symbols.linux.extensions.bash*), 279  
`hive_offset()` (*RegistryHive* property), 185  
`HiveGenerator` (class in *volatility.plugins.windows.registry.hivelist*), 470  
`HiveList` (class in *volatility.plugins.windows.registry.hivelist*), 470  
`HiveScan` (class in *volatility.plugins.windows.registry.hivescan*), 472  
`HMAP_ENTRY` (class in *volatility.framework.symbols.windows.extensions.registry*), 367  
`HMAP_ENTRY.VolTemplateProxy` (class in *volatility.framework.symbols.windows.extensions.registry*), 367  
`Ifconfig` (class in *volatility.plugins.mac.ifconfig*), 439  
`ifnet` (class in *volatility.framework.symbols.mac.extensions*), 293



`ifnet.VolTemplateProxy` (class in `volatility.framework.symbols.mac.extensions`), 293  
`imag` (*Bin attribute*), 241  
`imag` (*BitField attribute*), 200  
`imag` (*Boolean attribute*), 202  
`imag` (*Char attribute*), 209  
`imag` (*Enumeration attribute*), 213  
`imag` (*Float attribute*), 216  
`imag` (*Hex attribute*), 242  
`imag` (*Integer attribute*), 220  
`imag` (*Pointer attribute*), 222  
`IMAGE_DOS_HEADER` (class in `volatility.framework.symbols.windows.extensions.pe`), 348  
`IMAGE_DOS_HEADER.VolTemplateProxy` (class in `volatility.framework.symbols.windows.extensions.pe`), 348  
`IMAGE_NT_HEADERS` (class in `volatility.framework.symbols.windows.extensions.pe`), 350  
`IMAGE_NT_HEADERS.VolTemplateProxy` (class in `volatility.framework.symbols.windows.extensions.pe`), 350  
`import_files()` (in module `volatility.framework`), 44  
`index()` (*Array method*), 198  
`index()` (*Bytes method*), 205  
`index()` (*Column method*), 129  
`index()` (*DataFormatInfo method*), 211  
`index()` (*HexBytes method*), 243  
`index()` (*MultiTypeData method*), 246  
`index()` (*String method*), 226  
`index()` (*TreeNode method*), 132, 239  
`inet_ntop()` (in module `volatility.framework.symbols.windows.extensions.networks`), 348  
`Info` (class in `volatility.plugins.windows.info`), 496  
`init_order_modules()` (*EPROCESS method*), 322  
`inpcb` (class in `volatility.framework.symbols.mac.extensions`), 295  
`inpcb.VolTemplateProxy` (class in `volatility.framework.symbols.mac.extensions`), 295  
`instance_type` (*BooleanRequirement attribute*), 70  
`instance_type` (*BytesRequirement attribute*), 72  
`instance_type` (*IntRequirement attribute*), 76  
`instance_type` (*SimpleTypeRequirement attribute*), 111  
`instance_type` (*StringRequirement attribute*), 84  
`instance_type` (*URIRequirement attribute*), 89  
`Integer` (class in `volatility.framework.objects`), 218  
`Integer.VolTemplateProxy` (class in `volatility.framework.objects`), 218  
`Intel` (class in `volatility.framework.layers.intel`), 150  
`Intel32e` (class in `volatility.framework.layers.intel`), 152  
`IntelPAE` (class in `volatility.framework.layers.intel`), 154  
`interface_version()` (in module `volatility.framework`), 44  
`IntermediateSymbolTable` (class in `volatility.framework.symbols.intermed`), 380  
`IntRequirement` (class in `volatility.framework.configuration.requirements`), 76  
`invalid()` (*HiveGenerator property*), 470  
`InvalidAddressException`, 406  
`invalidate_caches()` (*WarningFindSpec method*), 27  
`is_ancestor()` (*TreeGrid method*), 131, 238  
`is_free_pool()` (*POOL\_HEADER method*), 356  
`is_free_pool()` (*POOL\_HEADER\_VISTA method*), 358  
`is_integer()` (*Float method*), 216  
`is_nonpaged_pool()` (*POOL\_HEADER method*), 357  
`is_nonpaged_pool()` (*POOL\_HEADER\_VISTA method*), 358  
`is_paged_pool()` (*POOL\_HEADER method*), 357  
`is_paged_pool()` (*POOL\_HEADER\_VISTA method*), 358  
`is_readable()` (*Pointer method*), 222  
`is_suspicious()` (*vm\_area\_struct method*), 278  
`is_suspicious()` (*vm\_map\_entry method*), 309  
`is_vad_empty()` (*Malfind class method*), 499  
`is_valid()` (*BufferDataLayer method*), 177  
`is_valid()` (*CMHIVE method*), 362  
`is_valid()` (*CONTROL\_AREA method*), 317  
`is_valid()` (*DataLayerInterface method*), 117  
`is_valid()` (*DRIVER\_OBJECT method*), 320  
`is_valid()` (*elf method*), 282  
`is_valid()` (*Elf64Layer method*), 147  
`is_valid()` (*EPROCESS method*), 322  
`is_valid()` (*FILE\_OBJECT method*), 327  
`is_valid()` (*FileLayer method*), 179  
`is_valid()` (*hist\_entry method*), 280  
`is_valid()` (*Intel method*), 150  
`is_valid()` (*Intel32e method*), 153  
`is_valid()` (*IntelPAE method*), 155  
`is_valid()` (*KMUTANT method*), 329  
`is_valid()` (*LimeLayer method*), 167  
`is_valid()` (*LinearlyMappedLayer method*), 169  
`is_valid()` (*NonLinearlySegmentedLayer method*), 189  
`is_valid()` (*OBJECT\_HEADER method*), 355  
`is_valid()` (*OBJECT\_SYMBOLIC\_LINK method*), 339  
`is_valid()` (*PdbMSFStream method*), 172

`is_valid()` (*PdbMultiStreamFormat method*), 174  
`is_valid()` (*POOL\_TRACKER\_BIG\_PAGES method*), 360  
`is_valid()` (*QemuSuspendLayer method*), 182  
`is_valid()` (*RegistryHive method*), 185  
`is_valid()` (*SegmentedLayer method*), 191  
`is_valid()` (*SERVICE\_HEADER method*), 371  
`is_valid()` (*SERVICE\_RECORD method*), 373  
`is_valid()` (*SHARED\_CACHE\_MAP method*), 341  
`is_valid()` (*TranslationLayerInterface method*), 121  
`is_valid()` (*vfsmount method*), 276  
`is_valid()` (*VmwareLayer method*), 194  
`is_valid()` (*WindowsCrashDump32Layer method*), 142  
`is_valid()` (*WindowsCrashDump64Layer method*), 145  
`is_valid()` (*WindowsIntel method*), 157  
`is_valid()` (*WindowsIntel32e method*), 159  
`is_valid()` (*WindowsIntelPAE method*), 162  
`is_valid()` (*WindowsMixin method*), 164  
`is_valid_choice()` (*Enumeration property*), 214  
`isalnum()` (*Bytes method*), 205  
`isalnum()` (*HexBytes method*), 243  
`isalnum()` (*MultiTypeData method*), 247  
`isalnum()` (*String method*), 226  
`isalpha()` (*Bytes method*), 205  
`isalpha()` (*HexBytes method*), 243  
`isalpha()` (*MultiTypeData method*), 247  
`isalpha()` (*String method*), 226  
`isascii()` (*Bytes method*), 205  
`isascii()` (*HexBytes method*), 243  
`isascii()` (*MultiTypeData method*), 247  
`isascii()` (*String method*), 226  
`isatty()` (*FileHandlerInterface method*), 126  
`isatty()` (*NullFileHandler method*), 30  
`isdecimal()` (*String method*), 226  
`isdigit()` (*Bytes method*), 205  
`isdigit()` (*HexBytes method*), 243  
`isdigit()` (*MultiTypeData method*), 247  
`isdigit()` (*String method*), 227  
`ISF_EXTENSIONS` (*in module volatility.framework.constants*), 92  
`ISF_MINIMUM_DEPRECATED` (*in module volatility.framework.constants*), 92  
`ISF_MINIMUM_SUPPORTED` (*in module volatility.framework.constants*), 92  
`IsfInfo` (*class in volatility.plugins.isfinfo*), 538  
`ISFormatTable` (*class in volatility.framework.symbols.intermed*), 377  
`isidentifier()` (*String method*), 227  
`islower()` (*Bytes method*), 205  
`islower()` (*HexBytes method*), 243  
`islower()` (*MultiTypeData method*), 247  
`islower()` (*String method*), 227  
`isnumeric()` (*String method*), 227  
`isprintable()` (*String method*), 227  
`isspace()` (*Bytes method*), 205  
`isspace()` (*HexBytes method*), 244  
`isspace()` (*MultiTypeData method*), 247  
`isspace()` (*String method*), 227  
`istitle()` (*Bytes method*), 205  
`istitle()` (*HexBytes method*), 244  
`istitle()` (*MultiTypeData method*), 247  
`istitle()` (*String method*), 227  
`isupper()` (*Bytes method*), 205  
`isupper()` (*HexBytes method*), 244  
`isupper()` (*MultiTypeData method*), 247  
`isupper()` (*String method*), 227  
`items()` (*HierarchicalDict method*), 108  
`items()` (*LayerContainer method*), 119  
`items()` (*ObjectInformation method*), 123  
`items()` (*ReadOnlyMapping method*), 125  
`items()` (*SymbolSpace method*), 251  
`items()` (*SymbolSpaceInterface method*), 137

## J

`JarHandler` (*class in volatility.framework.layers.resources*), 187  
`join()` (*Bytes method*), 205  
`join()` (*HexBytes method*), 244  
`join()` (*MultiTypeData method*), 247  
`join()` (*String method*), 227  
`JsonLinesRenderer` (*class in volatility.cli.text\_renderer*), 41  
`JsonRenderer` (*class in volatility.cli.text\_renderer*), 41

## K

`Kauth_listeners` (*class in volatility.plugins.mac.kauth\_listeners*), 441  
`kauth_scope` (*class in volatility.framework.symbols.mac.extensions*), 296  
`kauth_scope.VolTemplateProxy` (*class in volatility.framework.symbols.mac.extensions*), 296  
`Kauth_scopes` (*class in volatility.plugins.mac.kauth\_scopes*), 442  
`KDDEBUGGER_DATA64` (*class in volatility.framework.symbols.windows.extensions.kdbg*), 346  
`KDDEBUGGER_DATA64.VolTemplateProxy` (*class in volatility.framework.symbols.windows.extensions.kdbg*), 346  
`KERNEL_MODULE_NAMES` (*in module volatility.framework.constants.windows*), 93  
`KernelPDBScanner` (*class in volatility.framework.automagic.pdbscan*), 54

- Kevents (class in *volatility.plugins.mac.kevents*), 444  
 KEY\_COMP\_NAME (RegKeyFlags attribute), 369  
 KEY\_HIVE\_ENTRY (RegKeyFlags attribute), 369  
 KEY\_HIVE\_EXIT (RegKeyFlags attribute), 369  
 KEY\_IS\_VOLATILE (RegKeyFlags attribute), 369  
 key\_iterator() (PrintKey class method), 474  
 KEY\_NO\_DELETE (RegKeyFlags attribute), 369  
 KEY\_PREFEF\_HANDLE (RegKeyFlags attribute), 369  
 KEY\_SYM\_LINK (RegKeyFlags attribute), 369  
 KEY\_VIRT\_MIRRORED (RegKeyFlags attribute), 369  
 KEY\_VIRT\_TARGET (RegKeyFlags attribute), 369  
 KEY\_VIRTUAL\_STORE (RegKeyFlags attribute), 369  
 Keyboard\_notifiers (class in *volatility.plugins.linux.keyboard\_notifiers*), 420  
 keys() (HierarchicalDict method), 108  
 keys() (LayerContainer method), 119  
 keys() (ObjectInformation method), 123  
 keys() (ReadOnlyMapping method), 125  
 keys() (SymbolSpace method), 251  
 keys() (SymbolSpaceInterface method), 137  
 KMUTANT (class in *volatility.framework.symbols.windows.extensions*), 327  
 KMUTANT.VolTemplateProxy (class in *volatility.framework.symbols.windows.extensions*), 327  
 kobject (class in *volatility.framework.symbols.linux.extensions*), 261  
 kobject.VolTemplateProxy (class in *volatility.framework.symbols.linux.extensions*), 261  
 KSYSTEM\_TIME (class in *volatility.framework.symbols.windows.extensions*), 329  
 KSYSTEM\_TIME.VolTemplateProxy (class in *volatility.framework.symbols.windows.extensions*), 329  
 KTHREAD (class in *volatility.framework.symbols.windows.extensions*), 330  
 KTHREAD.VolTemplateProxy (class in *volatility.framework.symbols.windows.extensions*), 331
- L**
- layer\_name() (BytesScanner property), 140  
 layer\_name() (Module property), 95  
 layer\_name() (ModuleInterface property), 115  
 layer\_name() (MultiStringScanner property), 140  
 layer\_name() (PageMapScanner property), 66  
 layer\_name() (PdbSignatureScanner property), 376  
 layer\_name() (PoolHeaderScanner property), 511  
 layer\_name() (RegexScanner property), 140  
 layer\_name() (ScannerInterface property), 120  
 layer\_name() (SizedModule property), 97  
 LayerContainer (class in *volatility.framework.interfaces.layers*), 118  
 LayerException, 406  
 LayerListRequirement (class in *volatility.framework.configuration.requirements*), 77  
 layers() (Context property), 94  
 layers() (ContextInterface property), 113  
 LayerStacker (class in *volatility.framework.automagic.stacker*), 58  
 LayerWriter (class in *volatility.plugins.layerwriter*), 540  
 length() (DataFormatInfo property), 211  
 LimeFormatException, 166  
 LimeLayer (class in *volatility.framework.layers.lime*), 166  
 LimeStacker (class in *volatility.framework.layers.lime*), 168  
 LinearlyMappedLayer (class in *volatility.framework.layers.linear*), 169  
 LINUX\_BANNERS\_PATH (in module *volatility.framework.constants*), 92  
 LinuxBannerCache (class in *volatility.framework.automagic.linux*), 47  
 LinuxIntelStacker (class in *volatility.framework.automagic.linux*), 49  
 LinuxKernelIntermedSymbols (class in *volatility.framework.symbols.linux*), 253  
 LinuxMetadata (class in *volatility.framework.symbols.metadata*), 404  
 LinuxSymbolFinder (class in *volatility.framework.automagic.linux*), 49  
 LinuxUtilities (class in *volatility.framework.symbols.linux*), 256  
 list\_all\_isf\_files() (IsfInfo class method), 538  
 list\_big\_pools() (BigPools class method), 478  
 LIST\_ENTRY (class in *volatility.framework.symbols.windows.extensions*), 332  
 LIST\_ENTRY.VolTemplateProxy (class in *volatility.framework.symbols.windows.extensions*), 332  
 List\_Files (class in *volatility.plugins.mac.list\_files*), 446  
 list\_files() (List\_Files class method), 446  
 list\_handlers (ResourceAccessor attribute), 187  
 list\_head (class in *volatility.framework.symbols.linux.extensions*), 263  
 list\_head.VolTemplateProxy (class in *volatility.framework.symbols.linux.extensions*), 263  
 list\_hive\_objects() (HiveList class method), 470

- `list_hives()` (*HiveList class method*), 471
  - `list_injections()` (*Malfind class method*), 499
  - `list_kauth_scopes()` (*Kauth\_scopes class method*), 443
  - `list_kernel_events()` (*Kevents class method*), 444
  - `list_modules()` (*Lsmmod class method*), 422, 448
  - `list_modules()` (*Modules class method*), 506
  - `list_mounts()` (*Mount class method*), 452
  - `list_plugins()` (*in module volatility.framework*), 44
  - `list_processes()` (*PsList class method*), 517
  - `list_processes()` (*Volshell method*), 40
  - `list_sockets()` (*Netstat class method*), 454
  - `list_tasks()` (*PsList class method*), 428
  - `list_tasks()` (*PsTree class method*), 430
  - `list_tasks()` (*Volshell method*), 35, 37
  - `list_tasks_allproc()` (*PsList class method*), 459
  - `list_tasks_pid_hash_table()` (*PsList class method*), 460
  - `list_tasks_process_group()` (*PsList class method*), 460
  - `list_tasks_sessions()` (*PsList class method*), 460
  - `list_tasks_tasks()` (*PsList class method*), 461
  - `list_userassist()` (*UserAssist method*), 476
  - `list_vads()` (*VadInfo class method*), 528
  - `ListRequirement` (*class in volatility.framework.configuration.requirements*), 79
  - `ljust()` (*Bytes method*), 205
  - `ljust()` (*HexBytes method*), 244
  - `ljust()` (*MultiTypeData method*), 247
  - `ljust()` (*String method*), 227
  - `load_banners()` (*LinuxBannerCache class method*), 48
  - `load_banners()` (*MacBannerCache class method*), 52
  - `load_banners()` (*SymbolBannerCache class method*), 61
  - `load_cached_validations()` (*in module volatility.schemas*), 543
  - `load_file()` (*Volshell method*), 33, 35, 37, 40
  - `load_order_modules()` (*EPROCESS method*), 322
  - `load_pdb_layer()` (*PdbReader class method*), 374
  - `load_windows_symbol_table()` (*PDBUtility class method*), 375
  - `locate_banners()` (*Banners class method*), 534
  - `location()` (*FileLayer property*), 179
  - `LOGLEVEL_V` (*in module volatility.framework.constants*), 92
  - `LOGLEVEL_VV` (*in module volatility.framework.constants*), 92
  - `LOGLEVEL_VVV` (*in module volatility.framework.constants*), 92
  - `LOGLEVEL_VVVV` (*in module volatility.framework.constants*), 92
  - `lookup()` (*Enumeration method*), 214
  - `lookup()` (*Enumeration.VolTemplateProxy class method*), 212
  - `lookup_module_address()` (*LinuxUtilities class method*), 256
  - `lookup_module_address()` (*MacUtilities class method*), 291
  - `lookup_user_sids()` (*GetSIDs method*), 493
  - `lower()` (*Bytes method*), 206
  - `lower()` (*HexBytes method*), 244
  - `lower()` (*MultiTypeData method*), 247
  - `lower()` (*String method*), 227
  - `Lsmmod` (*class in volatility.plugins.linux.lsmmod*), 421
  - `Lsmmod` (*class in volatility.plugins.mac.lsmmod*), 447
  - `Lsof` (*class in volatility.plugins.linux.lsof*), 423
  - `Lsof` (*class in volatility.plugins.mac.lsof*), 449
  - `lstrip()` (*Bytes method*), 206
  - `lstrip()` (*HexBytes method*), 244
  - `lstrip()` (*MultiTypeData method*), 247
  - `lstrip()` (*String method*), 228
- ## M
- `MAC_BANNERS_PATH` (*in module volatility.framework.constants*), 92
  - `MacBannerCache` (*class in volatility.framework.automagic.mac*), 51
  - `MacIntelStacker` (*class in volatility.framework.automagic.mac*), 52
  - `MacKernelIntermedSymbols` (*class in volatility.framework.symbols.mac*), 288
  - `MacSymbolFinder` (*class in volatility.framework.automagic.mac*), 53
  - `MacUtilities` (*class in volatility.framework.symbols.mac*), 291
  - `MAGIC` (*Elf64Layer attribute*), 147
  - `MAGIC` (*LimeLayer attribute*), 166
  - `main()` (*in module volatility.cli*), 29
  - `main()` (*in module volatility.cli.volshell*), 30
  - `major()` (*super\_block property*), 273
  - `make_subconfig()` (*AutomagicInterface class method*), 100
  - `make_subconfig()` (*Banners class method*), 534
  - `make_subconfig()` (*Bash class method*), 409, 434
  - `make_subconfig()` (*BashIntermedSymbols class method*), 287
  - `make_subconfig()` (*BigPools class method*), 478
  - `make_subconfig()` (*BufferDataLayer class method*), 177
  - `make_subconfig()` (*Check\_afinfo class method*), 411
  - `make_subconfig()` (*Check\_creds class method*), 412
  - `make_subconfig()` (*Check\_idt class method*), 414



`make_subconfig()` (*Check\_modules class method*), 415  
`make_subconfig()` (*Check\_syscall class method*), 417, 435  
`make_subconfig()` (*Check\_sysctl class method*), 437  
`make_subconfig()` (*Check\_trap\_table class method*), 438  
`make_subconfig()` (*CmdLine class method*), 480  
`make_subconfig()` (*ConfigurableInterface class method*), 103  
`make_subconfig()` (*ConfigWriter class method*), 535  
`make_subconfig()` (*ConstructionMagic class method*), 47  
`make_subconfig()` (*DataLayerInterface class method*), 117  
`make_subconfig()` (*DllList class method*), 482  
`make_subconfig()` (*DriverIrp class method*), 483  
`make_subconfig()` (*DriverScan class method*), 485  
`make_subconfig()` (*DumpFiles class method*), 487  
`make_subconfig()` (*Elf64Layer class method*), 148  
`make_subconfig()` (*Elfs class method*), 419  
`make_subconfig()` (*Envvars class method*), 489  
`make_subconfig()` (*FileLayer class method*), 179  
`make_subconfig()` (*FileScan class method*), 490  
`make_subconfig()` (*FrameworkInfo class method*), 537  
`make_subconfig()` (*GetServiceSIDs class method*), 492  
`make_subconfig()` (*GetSIDs class method*), 493  
`make_subconfig()` (*Handles class method*), 495  
`make_subconfig()` (*HiveList class method*), 471  
`make_subconfig()` (*HiveScan class method*), 473  
`make_subconfig()` (*Ifconfig class method*), 440  
`make_subconfig()` (*Info class method*), 498  
`make_subconfig()` (*Intel class method*), 151  
`make_subconfig()` (*Intel32e class method*), 153  
`make_subconfig()` (*IntelPAE class method*), 155  
`make_subconfig()` (*IntermediateSymbolTable class method*), 382  
`make_subconfig()` (*IsfInfo class method*), 539  
`make_subconfig()` (*ISFormatTable class method*), 379  
`make_subconfig()` (*Kauth\_listeners class method*), 441  
`make_subconfig()` (*Kauth\_scopes class method*), 443  
`make_subconfig()` (*KernelPDBScanner class method*), 56  
`make_subconfig()` (*Kevents class method*), 445  
`make_subconfig()` (*Keyboard\_notifiers class method*), 420  
`make_subconfig()` (*LayerStacker class method*), 59  
`make_subconfig()` (*LayerWriter class method*), 540  
`make_subconfig()` (*LimeLayer class method*), 167  
`make_subconfig()` (*LinearlyMappedLayer class method*), 170  
`make_subconfig()` (*LinuxBannerCache class method*), 48  
`make_subconfig()` (*LinuxKernelIntermedSymbols class method*), 255  
`make_subconfig()` (*LinuxSymbolFinder class method*), 50  
`make_subconfig()` (*List\_Files class method*), 446  
`make_subconfig()` (*Lsmmod class method*), 422, 448  
`make_subconfig()` (*Lsof class method*), 423, 449  
`make_subconfig()` (*MacBannerCache class method*), 52  
`make_subconfig()` (*MacKernelIntermedSymbols class method*), 290  
`make_subconfig()` (*MacSymbolFinder class method*), 54  
`make_subconfig()` (*Malfind class method*), 425, 451, 500  
`make_subconfig()` (*Maps class method*), 426, 456  
`make_subconfig()` (*Memmap class method*), 501  
`make_subconfig()` (*ModScan class method*), 503  
`make_subconfig()` (*Modules class method*), 506  
`make_subconfig()` (*Mount class method*), 453  
`make_subconfig()` (*MutantScan class method*), 507  
`make_subconfig()` (*NetScan class method*), 510  
`make_subconfig()` (*Netstat class method*), 454  
`make_subconfig()` (*NonLinearlySegmentedLayer class method*), 189  
`make_subconfig()` (*PdbMSFStream class method*), 172  
`make_subconfig()` (*PdbMultiStreamFormat class method*), 175  
`make_subconfig()` (*PluginInterface class method*), 128  
`make_subconfig()` (*PoolScanner class method*), 513  
`make_subconfig()` (*PrintKey class method*), 475  
`make_subconfig()` (*Privs class method*), 515  
`make_subconfig()` (*Psaux class method*), 457  
`make_subconfig()` (*PsList class method*), 428, 461, 517  
`make_subconfig()` (*PsScan class method*), 519  
`make_subconfig()` (*PsTree class method*), 430, 462, 521  
`make_subconfig()` (*QemuSuspendLayer class method*), 182  
`make_subconfig()` (*RegistryHive class method*), 185  
`make_subconfig()` (*SegmentedLayer class method*), 191  
`make_subconfig()` (*Socket\_filters class method*), 464  
`make_subconfig()` (*SSDT class method*), 523  
`make_subconfig()` (*Strings class method*), 525

`make_subconfig()` (*SymbolBannerCache class method*), 61

`make_subconfig()` (*SymbolFinder class method*), 63

`make_subconfig()` (*SymbolTableInterface class method*), 139

`make_subconfig()` (*SymlinkScan class method*), 526

`make_subconfig()` (*Timeliner class method*), 542

`make_subconfig()` (*Timers class method*), 465

`make_subconfig()` (*TranslationLayerInterface class method*), 121

`make_subconfig()` (*Trustedbsd class method*), 467

`make_subconfig()` (*tty\_check class method*), 432

`make_subconfig()` (*UserAssist class method*), 476

`make_subconfig()` (*VadInfo class method*), 528

`make_subconfig()` (*VerInfo class method*), 531

`make_subconfig()` (*Version1Format class method*), 385

`make_subconfig()` (*Version2Format class method*), 387

`make_subconfig()` (*Version3Format class method*), 390

`make_subconfig()` (*Version4Format class method*), 392

`make_subconfig()` (*Version5Format class method*), 395

`make_subconfig()` (*Version6Format class method*), 398

`make_subconfig()` (*Version7Format class method*), 400

`make_subconfig()` (*Version8Format class method*), 403

`make_subconfig()` (*VFSevents class method*), 468

`make_subconfig()` (*VirtMap class method*), 532

`make_subconfig()` (*VmwareLayer class method*), 194

`make_subconfig()` (*Volshell class method*), 33, 35, 37, 40

`make_subconfig()` (*WindowsCrashDump32Layer class method*), 142

`make_subconfig()` (*WindowsCrashDump64Layer class method*), 145

`make_subconfig()` (*WindowsIntel class method*), 157

`make_subconfig()` (*WindowsIntel32e class method*), 159

`make_subconfig()` (*WindowsIntelPAE class method*), 162

`make_subconfig()` (*WindowsKernelIntermedSymbols class method*), 314

`make_subconfig()` (*WindowsMixin class method*), 164

`make_subconfig()` (*WinSwapLayers class method*), 67

`make_subconfig()` (*WintelHelper class method*), 69

`maketrans()` (*Bytes static method*), 206

`maketrans()` (*HexBytes static method*), 244

`maketrans()` (*MultiTypeData static method*), 247

`maketrans()` (*String static method*), 228

`Malfind` (*class in volatility.plugins.linux.malfind*), 424

`Malfind` (*class in volatility.plugins.mac.malfind*), 450

`Malfind` (*class in volatility.plugins.windows.malfind*), 499

`mapping()` (*Elf64Layer method*), 148

`mapping()` (*Intel method*), 151

`mapping()` (*Intel32e method*), 153

`mapping()` (*IntelPAE method*), 155

`mapping()` (*LimeLayer method*), 167

`mapping()` (*LinearlyMappedLayer method*), 170

`mapping()` (*NonLinearlySegmentedLayer method*), 189

`mapping()` (*PdbMSFStream method*), 172

`mapping()` (*PdbMultiStreamFormat method*), 175

`mapping()` (*QemuSuspendLayer method*), 183

`mapping()` (*RegistryHive method*), 185

`mapping()` (*SegmentedLayer method*), 191

`mapping()` (*TranslationLayerInterface method*), 121

`mapping()` (*VmwareLayer method*), 194

`mapping()` (*WindowsCrashDump32Layer method*), 142

`mapping()` (*WindowsCrashDump64Layer method*), 145

`mapping()` (*WindowsIntel method*), 157

`mapping()` (*WindowsIntel32e method*), 160

`mapping()` (*WindowsIntelPAE method*), 162

`mapping()` (*WindowsMixin method*), 164

`Maps` (*class in volatility.plugins.linux.proc*), 426

`Maps` (*class in volatility.plugins.mac.proc\_maps*), 455

`mask_mods_list()` (*LinuxUtilities class method*), 256

`mask_mods_list()` (*MacUtilities class method*), 291

`max_depth()` (*TreeGrid method*), 131, 238

`max_pdb_size` (*KernelPDBScanner attribute*), 57

`maximum_address` (*Intel attribute*), 151

`maximum_address` (*Intel32e attribute*), 153

`maximum_address` (*IntelPAE attribute*), 155

`maximum_address` (*WindowsIntel attribute*), 158

`maximum_address` (*WindowsIntel32e attribute*), 160

`maximum_address` (*WindowsIntelPAE attribute*), 162

`maximum_address` (*WindowsMixin attribute*), 164

`maximum_address()` (*BufferDataLayer property*), 177

`maximum_address()` (*DataLayerInterface property*), 117

`maximum_address()` (*Elf64Layer property*), 148

`maximum_address()` (*FileLayer property*), 180

`maximum_address()` (*LimeLayer property*), 167

`maximum_address()` (*LinearlyMappedLayer property*), 170

- `maximum_address()` (*NonLinearlySegmentedLayer property*), 189
- `maximum_address()` (*PdbMSFStream property*), 172
- `maximum_address()` (*PdbMultiStreamFormat property*), 175
- `maximum_address()` (*QemuSuspendLayer property*), 183
- `maximum_address()` (*RegistryHive property*), 186
- `maximum_address()` (*SegmentedLayer property*), 191
- `maximum_address()` (*TranslationLayerInterface property*), 121
- `maximum_address()` (*VmwareLayer property*), 194
- `maximum_address()` (*WindowsCrashDump32Layer property*), 142
- `maximum_address()` (*WindowsCrashDump64Layer property*), 145
- `MAXSIZE_DEFAULT` (*VadInfo attribute*), 528
- `mem_order_modules()` (*EPROCESS method*), 322
- `member()` (*AggregateType method*), 197
- `member()` (*ClassType method*), 211
- `member()` (*CM\_KEY\_BODY method*), 364
- `member()` (*CM\_KEY\_NODE method*), 365
- `member()` (*CM\_KEY\_VALUE method*), 367
- `member()` (*CMHIVE method*), 362
- `member()` (*CONTROL\_AREA method*), 317
- `member()` (*dentry method*), 258
- `member()` (*DEVICE\_OBJECT method*), 318
- `member()` (*DRIVER\_OBJECT method*), 320
- `member()` (*elf method*), 282
- `member()` (*elf\_phdr method*), 283
- `member()` (*elf\_sym method*), 285
- `member()` (*EPROCESS method*), 322
- `member()` (*ETHREAD method*), 324
- `member()` (*EX\_FAST\_REF method*), 325
- `member()` (*FILE\_OBJECT method*), 327
- `member()` (*fileglob method*), 293
- `member()` (*files\_struct method*), 259
- `member()` (*fs\_struct method*), 261
- `member()` (*GenericIntelProcess method*), 253
- `member()` (*hist\_entry method*), 280
- `member()` (*HMAP\_ENTRY method*), 368
- `member()` (*ifnet method*), 294
- `member()` (*IMAGE\_DOS\_HEADER method*), 349
- `member()` (*IMAGE\_NT\_HEADERS method*), 351
- `member()` (*inpcb method*), 296
- `member()` (*kauth\_scope method*), 297
- `member()` (*KDDEBUGGER\_DATA64 method*), 347
- `member()` (*KMUTANT method*), 329
- `member()` (*kobject method*), 262
- `member()` (*KSYSTEM\_TIME method*), 330
- `member()` (*KTHREAD method*), 332
- `member()` (*LIST\_ENTRY method*), 333
- `member()` (*list\_head method*), 264
- `member()` (*mm\_struct method*), 265
- `member()` (*MMVAD method*), 335
- `member()` (*MMVAD\_SHORT method*), 337
- `member()` (*module method*), 267
- `member()` (*mount method*), 269
- `member()` (*OBJECT\_HEADER method*), 355
- `member()` (*OBJECT\_SYMBOLIC\_LINK method*), 339
- `member()` (*POOL\_HEADER method*), 357
- `member()` (*POOL\_HEADER\_VISTA method*), 358
- `member()` (*POOL\_TRACKER\_BIG\_PAGES method*), 360
- `member()` (*proc method*), 299
- `member()` (*qstr method*), 270
- `member()` (*queue\_entry method*), 301
- `member()` (*SERVICE\_HEADER method*), 371
- `member()` (*SERVICE\_RECORD method*), 373
- `member()` (*SHARED\_CACHE\_MAP method*), 341
- `member()` (*sockaddr method*), 302
- `member()` (*sockaddr\_dl method*), 304
- `member()` (*socket method*), 305
- `member()` (*struct\_file method*), 272
- `member()` (*StructType method*), 231
- `member()` (*super\_block method*), 273
- `member()` (*sysctl\_oid method*), 307
- `member()` (*task\_struct method*), 275
- `member()` (*TOKEN method*), 342
- `member()` (*UNICODE\_STRING method*), 344
- `member()` (*UnionType method*), 232
- `member()` (*VACB method*), 346
- `member()` (*vfsmount method*), 276
- `member()` (*vm\_area\_struct method*), 278
- `member()` (*vm\_map\_entry method*), 309
- `member()` (*vm\_map\_object method*), 310
- `member()` (*vnode method*), 312
- `Memmap` (*class in volatility.plugins.windows.memmap*), 501
- `merge()` (*HierarchicalDict method*), 108
- `metadata()` (*BashIntermedSymbols property*), 287
- `metadata()` (*BufferDataLayer property*), 177
- `metadata()` (*DataLayerInterface property*), 117
- `metadata()` (*Elf64Layer property*), 148
- `metadata()` (*FileLayer property*), 180
- `metadata()` (*Intel property*), 151
- `metadata()` (*Intel32e property*), 153
- `metadata()` (*IntelPAE property*), 155
- `metadata()` (*IntermediateSymbolTable property*), 383
- `metadata()` (*ISFormatTable property*), 379
- `metadata()` (*LimeLayer property*), 167
- `metadata()` (*LinearlyMappedLayer property*), 170
- `metadata()` (*LinuxKernelIntermedSymbols property*), 255
- `metadata()` (*MacKernelIntermedSymbols property*), 290

- `metadata()` (*NonLinearlySegmentedLayer* property), 189
- `metadata()` (*PdbMSFStream* property), 173
- `metadata()` (*PdbMultiStreamFormat* property), 175
- `metadata()` (*QemuSuspendLayer* property), 183
- `metadata()` (*RegistryHive* property), 186
- `metadata()` (*SegmentedLayer* property), 191
- `metadata()` (*TranslationLayerInterface* property), 121
- `metadata()` (*Version1Format* property), 385
- `metadata()` (*Version2Format* property), 388
- `metadata()` (*Version3Format* property), 390
- `metadata()` (*Version4Format* property), 393
- `metadata()` (*Version5Format* property), 395
- `metadata()` (*Version6Format* property), 398
- `metadata()` (*Version7Format* property), 401
- `metadata()` (*Version8Format* property), 403
- `metadata()` (*VmwareLayer* property), 194
- `metadata()` (*WindowsCrashDump32Layer* property), 142
- `metadata()` (*WindowsCrashDump64Layer* property), 145
- `metadata()` (*WindowsIntel* property), 158
- `metadata()` (*WindowsIntel32e* property), 160
- `metadata()` (*WindowsIntelPAE* property), 162
- `metadata()` (*WindowsKernelIntermedSymbols* property), 314
- `metadata()` (*WindowsMixin* property), 164
- `MetadataInterface` (class in *volatility.framework.interfaces.symbols*), 134
- `method_fixed_mapping()` (*KernelPDBScanner* method), 57
- `method_kdbg_offset()` (*KernelPDBScanner* method), 57
- `method_module_offset()` (*KernelPDBScanner* method), 57
- `methods` (*KernelPDBScanner* attribute), 57
- `minimum_address` (*Intel* attribute), 151
- `minimum_address` (*Intel32e* attribute), 153
- `minimum_address` (*IntelPAE* attribute), 155
- `minimum_address` (*WindowsIntel* attribute), 158
- `minimum_address` (*WindowsIntel32e* attribute), 160
- `minimum_address` (*WindowsIntelPAE* attribute), 162
- `minimum_address` (*WindowsMixin* attribute), 164
- `minimum_address()` (*BufferDataLayer* property), 178
- `minimum_address()` (*DataLayerInterface* property), 117
- `minimum_address()` (*Elf64Layer* property), 148
- `minimum_address()` (*FileLayer* property), 180
- `minimum_address()` (*LimeLayer* property), 167
- `minimum_address()` (*LinearlyMappedLayer* property), 170
- `minimum_address()` (*NonLinearlySegmentedLayer* property), 189
- `minimum_address()` (*PdbMSFStream* property), 173
- `minimum_address()` (*PdbMultiStreamFormat* property), 175
- `minimum_address()` (*QemuSuspendLayer* property), 183
- `minimum_address()` (*RegistryHive* property), 186
- `minimum_address()` (*SegmentedLayer* property), 191
- `minimum_address()` (*TranslationLayerInterface* property), 121
- `minimum_address()` (*VmwareLayer* property), 194
- `minimum_address()` (*WindowsCrashDump32Layer* property), 143
- `minimum_address()` (*WindowsCrashDump64Layer* property), 145
- `minor()` (*super\_block* property), 273
- `MINORBITS` (*super\_block* attribute), 272
- `MissingModuleException`, 407
- `mm_struct` (class in *volatility.framework.symbols.linux.extensions*), 264
- `mm_struct.VolTemplateProxy` (class in *volatility.framework.symbols.linux.extensions*), 264
- `MMVAD` (class in *volatility.framework.symbols.windows.extensions*), 333
- `MMVAD.VolTemplateProxy` (class in *volatility.framework.symbols.windows.extensions*), 334
- `MMVAD_SHORT` (class in *volatility.framework.symbols.windows.extensions*), 335
- `MMVAD_SHORT.VolTemplateProxy` (class in *volatility.framework.symbols.windows.extensions*), 336
- `MODIFIED` (*TimeLinerType* attribute), 542
- `ModScan` (class in *volatility.plugins.windows.modscan*), 502
- `module`
  - volatility*, 27
  - volatility.cli*, 28
  - volatility.cli.text\_renderer*, 41
  - volatility.cli.volargparse*, 43
  - volatility.cli.volshell*, 29
  - volatility.cli.volshell.generic*, 30
  - volatility.cli.volshell.linux*, 34
  - volatility.cli.volshell.mac*, 36
  - volatility.cli.volshell.windows*, 38
  - volatility.framework*, 44
  - volatility.framework.automagic*, 45
  - volatility.framework.automagic.construct\_layers*, 46
  - volatility.framework.automagic.linux*,



[47](#)  
 volatility.framework.automagic.mac,  
[51](#)  
 volatility.framework.automagic.pdbscan,  
[54](#)  
 volatility.framework.automagic.stacker,  
[58](#)  
 volatility.framework.automagic.symbol\_cache,[187](#)  
[60](#)  
 volatility.framework.automagic.symbol\_finder,[140](#)  
[62](#)  
 volatility.framework.automagic.windows,  
[64](#)  
 volatility.framework.configuration,  
[70](#)  
 volatility.framework.configuration.requirements,  
[70](#)  
 volatility.framework.constants,[92](#)  
 volatility.framework.constants.linux,  
[93](#)  
 volatility.framework.constants.windows,  
[93](#)  
 volatility.framework.contexts,[93](#)  
 volatility.framework.exceptions,[406](#)  
 volatility.framework.interfaces,[99](#)  
 volatility.framework.interfaces.automagic,  
[99](#)  
 volatility.framework.interfaces.configuration,[101](#)  
[101](#)  
 volatility.framework.interfaces.context,  
[112](#)  
 volatility.framework.interfaces.layers,  
[116](#)  
 volatility.framework.interfaces.objects,  
[122](#)  
 volatility.framework.interfaces.plugins,  
[126](#)  
 volatility.framework.interfaces.renderers,  
[129](#)  
 volatility.framework.interfaces.symbols,  
[132](#)  
 volatility.framework.layers,[140](#)  
 volatility.framework.layers.codecs,  
[140](#)  
 volatility.framework.layers.crash,  
[141](#)  
 volatility.framework.layers.elf,[147](#)  
 volatility.framework.layers.intel,  
[150](#)  
 volatility.framework.layers.lime,  
[166](#)  
 volatility.framework.layers.linear,  
[169](#)  
 volatility.framework.layers.msf,[171](#)  
 volatility.framework.layers.physical,  
[176](#)  
 volatility.framework.layers.qemu,  
[181](#)  
 volatility.framework.layers.registry,  
[184](#)  
 volatility.framework.layers.resources,  
 volatility.framework.layers.scanners,  
 volatility.framework.layers.scanners.multiregex,  
[141](#)  
 volatility.framework.layers.segmented,  
[188](#)  
 volatility.framework.layers.vmware,  
[191](#)  
 volatility.framework.objects,[195](#)  
 volatility.framework.objects.templates,  
[234](#)  
 volatility.framework.objects.utility,  
[236](#)  
 volatility.framework.plugins,[236](#)  
 volatility.framework.renderers,[237](#)  
 volatility.framework.renderers.conversion,  
[240](#)  
 volatility.framework.renderers.format\_hints,  
[241](#)  
 volatility.framework.symbols,[249](#)  
 volatility.framework.symbols.generic,  
[252](#)  
 volatility.framework.symbols.intermed,  
[377](#)  
 volatility.framework.symbols.linux,  
[253](#)  
 volatility.framework.symbols.linux.bash,  
[285](#)  
 volatility.framework.symbols.linux.extensions,  
[257](#)  
 volatility.framework.symbols.linux.extensions.bash,  
[279](#)  
 volatility.framework.symbols.linux.extensions.c,  
[280](#)  
 volatility.framework.symbols.mac,  
[288](#)  
 volatility.framework.symbols.mac.extensions,  
[292](#)  
 volatility.framework.symbols.metadata,  
[404](#)  
 volatility.framework.symbols.native,  
[405](#)  
 volatility.framework.symbols.windows,  
[312](#)  
 volatility.framework.symbols.windows.extensions,  
[315](#)

volatility.framework.symbols.windows.extensions.kdbg, 346  
volatility.framework.symbols.windows.extensions.network, 348  
volatility.framework.symbols.windows.extensions.pe, 348  
volatility.framework.symbols.windows.extensions.pool, 352  
volatility.framework.symbols.windows.extensions.registry, 361  
volatility.framework.symbols.windows.extensions.syscall, 370  
volatility.framework.symbols.windows.pdbconv, 373  
volatility.framework.symbols.windows.pdbutil, 375  
volatility.framework.symbols.windows.version, 377  
volatility.framework.symbols.wrappers, 406  
volatility.plugins, 408  
volatility.plugins.banners, 533  
volatility.plugins.configwriter, 535  
volatility.plugins.frameworkinfo, 536  
volatility.plugins.isfinfo, 538  
volatility.plugins.layerwriter, 540  
volatility.plugins.linux, 408  
volatility.plugins.linux.bash, 409  
volatility.plugins.linux.check\_afinfo, 410  
volatility.plugins.linux.check\_creds, 412  
volatility.plugins.linux.check\_idt, 413  
volatility.plugins.linux.check\_modules, 415  
volatility.plugins.linux.check\_syscall, 416  
volatility.plugins.linux.elfs, 418  
volatility.plugins.linux.keyboard\_notifications, 420  
volatility.plugins.linux.lsmmod, 421  
volatility.plugins.linux.lsof, 423  
volatility.plugins.linux.malfind, 424  
volatility.plugins.linux.proc, 426  
volatility.plugins.linux.pslist, 427  
volatility.plugins.linux.pstree, 429  
volatility.plugins.linux.tty\_check, 431  
volatility.plugins.mac, 433  
volatility.plugins.mac.bash, 433  
volatility.plugins.mac.check\_syscall, 435  
volatility.plugins.mac.check\_sysctl, 436  
volatility.plugins.mac.check\_trap\_table, 438  
volatility.plugins.mac.ifconfig, 439  
volatility.plugins.mac.kauth\_listeners, 441  
volatility.plugins.mac.kauth\_scopes, 442  
volatility.plugins.mac.kevents, 444  
volatility.plugins.mac.list\_files, 446  
volatility.plugins.mac.lsmmod, 447  
volatility.plugins.mac.lsof, 449  
volatility.plugins.mac.malfind, 450  
volatility.plugins.mac.mount, 452  
volatility.plugins.mac.netstat, 454  
volatility.plugins.mac.proc\_maps, 455  
volatility.plugins.mac.psaux, 457  
volatility.plugins.mac.pslist, 459  
volatility.plugins.mac.pstree, 462  
volatility.plugins.mac.socket\_filters, 463  
volatility.plugins.mac.timers, 465  
volatility.plugins.mac.trustedbsd, 466  
volatility.plugins.mac.vfsevents, 468  
volatility.plugins.timeliner, 542  
volatility.plugins.windows, 469  
volatility.plugins.windows.bigpools, 477  
volatility.plugins.windows.cmdline, 479  
volatility.plugins.windows.dlllist, 481  
volatility.plugins.windows.driverirp, 483  
volatility.plugins.windows.driverscan, 484  
volatility.plugins.windows.dumpfiles, 486  
volatility.plugins.windows.envars, 488  
volatility.plugins.windows.filescan, 490  
volatility.plugins.windows.getservicesids, 491  
volatility.plugins.windows.getsids, 493  
volatility.plugins.windows.handles, 494

- volatility.plugins.windows.info, 496
  - volatility.plugins.windows.malfind, 499
  - volatility.plugins.windows.memmap, 501
  - volatility.plugins.windows.modscan, 502
  - volatility.plugins.windows.modules, 504
  - volatility.plugins.windows.mutantscan, 507
  - volatility.plugins.windows.netscan, 508
  - volatility.plugins.windows.poolscanner, 511
  - volatility.plugins.windows.privileges, 514
  - volatility.plugins.windows.pslist, 516
  - volatility.plugins.windows.psscan, 518
  - volatility.plugins.windows.pstree, 521
  - volatility.plugins.windows.registry, 469
  - volatility.plugins.windows.registry.hivelist, 470
  - volatility.plugins.windows.registry.hivescan, 472
  - volatility.plugins.windows.registry.printkey, 474
  - volatility.plugins.windows.registry.userscript, 476
  - volatility.plugins.windows.ssd, 522
  - volatility.plugins.windows.strings, 524
  - volatility.plugins.windows.symlinkscan, 526
  - volatility.plugins.windows.vadinfo, 528
  - volatility.plugins.windows.verinfo, 530
  - volatility.plugins.windows.virtmap, 532
  - volatility.schemas, 543
  - volatility.symbols, 544
  - Module (class in volatility.framework.contexts), 94
  - module (class in volatility.framework.symbols.linux.extensions), 266
  - module() (Context method), 94
  - module() (ContextInterface method), 113
  - module.VolTemplateProxy (class in volatility.framework.symbols.linux.extensions), 266
  - ModuleCollection (class in volatility.framework.contexts), 96
  - ModuleInterface (class in volatility.framework.interfaces.context), 114
  - Modules (class in volatility.plugins.windows.modules), 504
  - modules() (ModuleCollection property), 96
  - mount (class in volatility.framework.symbols.linux.extensions), 267
  - Mount (class in volatility.plugins.mac.mount), 452
  - mount.VolTemplateProxy (class in volatility.framework.symbols.linux.extensions), 268
  - MultiProcessing (Parallelism attribute), 92
  - MultiRegexp (class in volatility.framework.layers.scanners.multiregexp), 141
  - MultiRequirement (class in volatility.framework.configuration.requirements), 80
  - MultiStringScanner (class in volatility.framework.layers.scanners), 140
  - MultiTypeData (class in volatility.framework.renderers.format\_hints), 246
  - multitypedata\_as\_text() (in module volatility.cli.text\_renderer), 43
  - MutantScan (class in volatility.plugins.windows.mutantscan), 507
  - MuteProgress (class in volatility.cli), 29
- ## N
- name (CLIRenderer attribute), 41
  - name (CSVRenderer attribute), 41
  - name (JsonLinesRenderer attribute), 41
  - name (JsonRenderer attribute), 42
  - name (PrettyTextRenderer attribute), 42
  - name (QuickTextRenderer attribute), 42
  - name() (BooleanRequirement property), 70
  - name() (BufferDataLayer property), 178
  - name() (BytesRequirement property), 72
  - name() (ChoiceRequirement property), 73
  - name() (ClassRequirement property), 102
  - name() (CMHIVE property), 362
  - name() (Column property), 129
  - name() (ComplexListRequirement property), 75
  - name() (ConfigurableRequirementInterface property), 105
  - name() (ConstructableRequirementInterface property), 106
  - name() (DataLayerInterface property), 117
  - name() (Elf64Layer property), 148
  - name() (FileLayer property), 180
  - name() (Intel property), 151
  - name() (Intel32e property), 153

`name()` (*IntelPAE property*), 155  
`name()` (*IntRequirement property*), 76  
`name()` (*LayerListRequirement property*), 78  
`name()` (*LimeLayer property*), 167  
`name()` (*LinearlyMappedLayer property*), 170  
`name()` (*ListRequirement property*), 80  
`name()` (*Module property*), 95  
`name()` (*ModuleInterface property*), 115  
`name()` (*MultiRequirement property*), 81  
`name()` (*NonLinearlySegmentedLayer property*), 189  
`name()` (*PdbMSFStream property*), 173  
`name()` (*PdbMultiStreamFormat property*), 175  
`name()` (*PluginRequirement property*), 83  
`name()` (*QemuSuspendLayer property*), 183  
`name()` (*RegistryHive property*), 186  
`name()` (*RequirementInterface property*), 109  
`name()` (*SegmentedLayer property*), 192  
`name()` (*SimpleTypeRequirement property*), 111  
`name()` (*SizedModule property*), 97  
`name()` (*StringRequirement property*), 84  
`name()` (*SymbolInterface property*), 136  
`name()` (*SymbolTableRequirement property*), 86  
`name()` (*TranslationLayerInterface property*), 121  
`name()` (*TranslationLayerRequirement property*), 88  
`name()` (*URIRequirement property*), 89  
`name()` (*VersionRequirement property*), 91  
`name()` (*VmwareLayer property*), 194  
`name()` (*WindowsCrashDump32Layer property*), 143  
`name()` (*WindowsCrashDump64Layer property*), 145  
`name()` (*WindowsIntel property*), 158  
`name()` (*WindowsIntel32e property*), 160  
`name()` (*WindowsIntelPAE property*), 162  
`name()` (*WindowsMixin property*), 164  
`name_as_str()` (*qstr method*), 270  
`name_strip()` (*PdbReader method*), 374  
`NameInfo()` (*OBJECT\_HEADER property*), 353  
`natives()` (*BaseSymbolTableInterface property*), 133  
`natives()` (*BashIntermedSymbols property*), 287  
`natives()` (*IntermediateSymbolTable property*), 383  
`natives()` (*ISFormatTable property*), 379  
`natives()` (*LinuxKernelIntermedSymbols property*), 255  
`natives()` (*MacKernelIntermedSymbols property*), 290  
`natives()` (*NativeTable property*), 405  
`natives()` (*NativeTableInterface property*), 135  
`natives()` (*SymbolTableInterface property*), 139  
`natives()` (*Version1Format property*), 385  
`natives()` (*Version2Format property*), 388  
`natives()` (*Version3Format property*), 390  
`natives()` (*Version4Format property*), 393  
`natives()` (*Version5Format property*), 395  
`natives()` (*Version6Format property*), 398  
`natives()` (*Version7Format property*), 401

`natives()` (*Version8Format property*), 403  
`natives()` (*WindowsKernelIntermedSymbols property*), 314  
`NativeTable` (class in *volatility.framework.symbols.native*), 405  
`NativeTableInterface` (class in *volatility.framework.interfaces.symbols*), 134  
`NetScan` (class in *volatility.plugins.windows.netscan*), 508  
`Netstat` (class in *volatility.plugins.mac.netstat*), 454  
`new_requirement()` (*ComplexListRequirement method*), 75  
`new_requirement()` (*LayerListRequirement method*), 78  
`noninheritable` (class in *volatility.framework*), 44  
`NonLinearlySegmentedLayer` (class in *volatility.framework.layers.segmented*), 188  
`NONPAGED` (*PoolType attribute*), 514  
`NotApplicableValue` (class in *volatility.framework.renderers*), 237  
`NotAvailableValue` (class in *volatility.framework.renderers*), 237  
`NullFileHandler` (class in *volatility.cli.volshell.generic*), 30  
`num_syntab()` (*module property*), 267  
`numerator` (*Bin attribute*), 241  
`numerator` (*BitField attribute*), 200  
`numerator` (*Boolean attribute*), 202  
`numerator` (*Char attribute*), 209  
`numerator` (*Enumeration attribute*), 214  
`numerator` (*Hex attribute*), 242  
`numerator` (*Integer attribute*), 220  
`numerator` (*Pointer attribute*), 222

## O

`object()` (*Context method*), 94  
`object()` (*ContextInterface method*), 113  
`object()` (*Module method*), 95  
`object()` (*ModuleInterface method*), 115  
`object()` (*SizedModule method*), 98  
`object_from_symbol()` (*Module method*), 96  
`object_from_symbol()` (*ModuleInterface method*), 115  
`object_from_symbol()` (*SizedModule method*), 98  
`OBJECT_HEADER` (class in *volatility.framework.symbols.windows.extensions.pool*), 353  
`OBJECT_HEADER.VolTemplateProxy` (class in *volatility.framework.symbols.windows.extensions.pool*), 353  
`OBJECT_SYMBOLIC_LINK` (class in *volatility.framework.symbols.windows.extensions*), 337

- OBJECT\_SYMBOLIC\_LINK.VolTemplateProxy (class in volatility.framework.symbols.windows.extensions), 338
- ObjectInformation (class in volatility.framework.interfaces.objects), 122
- ObjectInterface (class in volatility.framework.interfaces.objects), 123
- ObjectInterface.VolTemplateProxy (class in volatility.framework.interfaces.objects), 123
- ObjectTemplate (class in volatility.framework.objects.templates), 234
- Off (Parallelism attribute), 92
- offset () (Module property), 96
- offset () (ModuleInterface property), 115
- offset () (SizedModule property), 98
- omap\_lookup () (PdbReader method), 374
- open () (Banners property), 534
- open () (Bash property), 410, 434
- open () (BigPools property), 478
- open () (Check\_afinfo property), 411
- open () (Check\_creds property), 413
- open () (Check\_idt property), 414
- open () (Check\_modules property), 416
- open () (Check\_syscall property), 417, 435
- open () (Check\_sysctl property), 437
- open () (Check\_trap\_table property), 438
- open () (CmdLine property), 480
- open () (ConfigWriter property), 536
- open () (DllList property), 482
- open () (DriverIrp property), 483
- open () (DriverScan property), 485
- open () (DumpFiles property), 487
- open () (Elfs property), 419
- open () (Envars property), 489
- open () (FileScan property), 490
- open () (FrameworkInfo property), 537
- open () (GetServiceSIDs property), 492
- open () (GetSIDs property), 494
- open () (Handles property), 496
- open () (HiveList property), 471
- open () (HiveScan property), 473
- open () (Ifconfig property), 440
- open () (Info property), 498
- open () (IsfInfo property), 539
- open () (Kauth\_listeners property), 441
- open () (Kauth\_scopes property), 443
- open () (Kevents property), 445
- open () (Keyboard\_notifiers property), 420
- open () (LayerWriter property), 540
- open () (List\_Files property), 446
- open () (Lsmode property), 422, 448
- open () (Lsof property), 424, 450
- open () (Malfind property), 425, 451, 500
- open () (Maps property), 427, 456
- open () (Memmap property), 501
- open () (ModScan property), 503
- open () (Modules property), 506
- open () (Mount property), 453
- open () (MutantScan property), 507
- open () (NetScan property), 510
- open () (Netstat property), 455
- open () (PluginInterface property), 128
- open () (PoolScanner property), 513
- open () (PrintKey property), 475
- open () (Privs property), 515
- open () (Psaux property), 458
- open () (PsList property), 429, 461, 517
- open () (PsScan property), 519
- open () (PsTree property), 430, 462, 521
- open () (ResourceAccessor method), 187
- open () (Socket\_filters property), 464
- open () (SSDT property), 523
- open () (Strings property), 525
- open () (SymlinkScan property), 527
- open () (Timeliner property), 543
- open () (Timers property), 465
- open () (Trustedbsd property), 467
- open () (tty\_check property), 432
- open () (UserAssist property), 476
- open () (VadInfo property), 529
- open () (VerInfo property), 531
- open () (VFSevents property), 468
- open () (VirtMap property), 533
- open () (Volshell property), 33, 35, 38, 40
- optional () (BooleanRequirement property), 71
- optional () (BytesRequirement property), 72
- optional () (ChoiceRequirement property), 73
- optional () (ClassRequirement property), 102
- optional () (ComplexListRequirement property), 75
- optional () (ConfigurableRequirementInterface property), 105
- optional () (ConstructableRequirementInterface property), 107
- optional () (in module volatility.cli.text\_renderer), 43
- optional () (IntRequirement property), 77
- optional () (LayerListRequirement property), 78
- optional () (ListRequirement property), 80
- optional () (MultiRequirement property), 81
- optional () (PluginRequirement property), 83
- optional () (RequirementInterface property), 110
- optional () (SimpleTypeRequirement property), 111
- optional () (StringRequirement property), 85
- optional () (SymbolTableRequirement property), 86
- optional () (TranslationLayerRequirement property), 88
- optional () (URIRequirement property), 89
- optional () (VersionRequirement property), 91



os (*LinuxBannerCache* attribute), 48  
os (*MacBannerCache* attribute), 52  
os (*SymbolBannerCache* attribute), 61  
OsDistinguisher (class in *volatility.framework.symbols.windows.versions*), 377  
output\_result() (*JsonLinesRenderer* method), 41  
output\_result() (*JsonRenderer* method), 42  
overlap (*PageMapScanner* attribute), 66  
overlap (*PdbSignatureScanner* attribute), 376  
owning\_process() (*ETHREAD* method), 324

## P

PACKAGE\_VERSION (in module *volatility.framework.constants*), 92  
PAGE\_MASK (*CONTROL\_AREA* attribute), 315  
PAGE\_SHIFT (in module *volatility.framework.constants.linux*), 93  
PAGE\_SIZE (*CONTROL\_AREA* attribute), 315  
page\_size (*Intel* attribute), 151  
page\_size (*Intel32e* attribute), 153  
page\_size (*IntelPAE* attribute), 156  
page\_size (*WindowsIntel* attribute), 158  
page\_size (*WindowsIntel32e* attribute), 160  
page\_size (*WindowsIntelPAE* attribute), 162  
page\_size (*WindowsMixin* attribute), 165  
page\_size() (*PdbMultiStreamFormat* property), 175  
PAGED (*PoolType* attribute), 514  
PagedInvalidAddressException, 407  
PageMapScanner (class in *volatility.framework.automagic.windows*), 66  
Parallelism (class in *volatility.framework.constants*), 92  
PARALLELISM (in module *volatility.framework.constants*), 92  
parent() (*TreeNode* property), 132, 239  
parent\_e\_type() (*elf\_phdr* property), 283  
parent\_offset() (*elf\_phdr* property), 283  
parent\_path() (in module *volatility.framework.interfaces.configuration*), 112  
parse\_args() (*HelpfulArgParser* method), 43  
parse\_intermixed\_args() (*HelpfulArgParser* method), 43  
parse\_known\_args() (*HelpfulArgParser* method), 43  
parse\_known\_intermixed\_args() (*HelpfulArgParser* method), 43  
parse\_string() (*PdbReader* static method), 374  
parse\_userassist\_data() (*UserAssist* method), 476  
partition() (*Bytes* method), 206  
partition() (*HexBytes* method), 244  
partition() (*MultiTypeData* method), 247  
partition() (*String* method), 228

path() (*dentry* method), 258  
path() (*TreeNode* property), 132, 239  
path\_changed() (*TreeNode* method), 132, 239  
path\_depth() (in module *volatility.framework.interfaces.configuration*), 112  
path\_depth() (*TreeGrid* static method), 131, 238  
path\_depth() (*TreeNode* property), 132, 239  
path\_for\_file() (*LinuxUtilities* class method), 256  
path\_head() (in module *volatility.framework.interfaces.configuration*), 112  
path\_join() (in module *volatility.framework.interfaces.configuration*), 112  
path\_sep (*TreeGrid* attribute), 238  
pdb\_age() (*WindowsMetadata* property), 404  
pdb\_guid() (*WindowsMetadata* property), 404  
pdb\_layer\_name() (*PdbReader* property), 374  
pdb\_symbol\_table() (*PdbMSFStream* property), 173  
pdb\_symbol\_table() (*PdbMultiStreamFormat* property), 175  
PDBFormatException, 171  
PdbMSFStream (class in *volatility.framework.layers.msf*), 171  
PdbMultiStreamFormat (class in *volatility.framework.layers.msf*), 174  
pdbname\_scan() (*PDBUtility* class method), 375  
PdbReader (class in *volatility.framework.symbols.windows.pdbconv*), 373  
PdbRetriever (class in *volatility.framework.symbols.windows.pdbconv*), 375  
PdbSignatureScanner (class in *volatility.framework.symbols.windows.pdbutil*), 376  
PDBUtility (class in *volatility.framework.symbols.windows.pdbutil*), 375  
pe\_version() (*WindowsMetadata* property), 404  
pe\_version\_string() (*WindowsMetadata* property), 404  
perm\_flags (*vm\_area\_struct* attribute), 278  
PHYSICAL\_DEFAULT (*PsList* attribute), 516  
PluginInterface (class in *volatility.framework.interfaces.plugins*), 127  
PluginRequirement (class in *volatility.framework.configuration.requirements*), 82  
PluginRequirementException, 407  
PLUGINS\_PATH (in module *volatility.framework.constants*), 92  
PluginVersionException, 407  
Pointer (class in *volatility.framework.objects*), 220  
Pointer.VolTemplateProxy (class in *volatility*

*ity.framework.objects*), 220  
 pointer\_to\_string() (in module *volatility.framework.objects.utility*), 236  
 POOL\_HEADER (class in *volatility.framework.symbols.windows.extensions.pool*), 355  
 POOL\_HEADER.VolTemplateProxy (class in *volatility.framework.symbols.windows.extensions.pool*), 355  
 POOL\_HEADER\_VISTA (class in *volatility.framework.symbols.windows.extensions.pool*), 357  
 POOL\_HEADER\_VISTA.VolTemplateProxy (class in *volatility.framework.symbols.windows.extensions.pool*), 357  
 pool\_scan() (*PoolScanner* class method), 513  
 POOL\_TRACKER\_BIG\_PAGES (class in *volatility.framework.symbols.windows.extensions.pool*), 359  
 POOL\_TRACKER\_BIG\_PAGES.VolTemplateProxy (class in *volatility.framework.symbols.windows.extensions.pool*), 359  
 pool\_type\_lookup(*POOL\_TRACKER\_BIG\_PAGES* attribute), 360  
 PoolConstraint (class in *volatility.plugins.windows.poolscanner*), 511  
 PoolHeaderScanner (class in *volatility.plugins.windows.poolscanner*), 511  
 PoolScanner (class in *volatility.plugins.windows.poolscanner*), 511  
 PoolType (class in *volatility.plugins.windows.poolscanner*), 514  
 populate() (*TreeGrid* method), 131, 238  
 populate\_config() (*CommandLine* method), 28  
 populate\_config() (*VolShell* method), 29  
 populate\_requirements\_argparse() (*CommandLine* method), 28  
 populate\_requirements\_argparse() (*VolShell* method), 29  
 populated() (*TreeGrid* property), 131, 238  
 possible\_architectures (*Disassembly* attribute), 130  
 preferred\_filename() (*FileHandlerInterface* property), 126  
 preferred\_filename() (*NullFileHandler* property), 30  
 preprocess() (*MultiRegex* method), 141  
 PrettyTextRenderer (class in *volatility.cli.text\_renderer*), 42  
 PrimitiveObject (class in *volatility.framework.objects*), 222  
 PrimitiveObject.VolTemplateProxy (class in *volatility.framework.objects*), 223  
 print\_help() (*HelpfulArgParser* method), 44  
 print\_usage() (*HelpfulArgParser* method), 44  
 PrintedProgress (class in *volatility.cli*), 29  
 PrintKey (class in *volatility.plugins.windows.registry.printkey*), 474  
 priority (*AutomagicInterface* attribute), 100  
 priority (*ConstructionMagic* attribute), 47  
 priority (*KernelPDBScanner* attribute), 57  
 priority (*LayerStacker* attribute), 59  
 priority (*LinuxBannerCache* attribute), 48  
 priority (*LinuxSymbolFinder* attribute), 50  
 priority (*MacBannerCache* attribute), 52  
 priority (*MacSymbolFinder* attribute), 54  
 priority (*SymbolBannerCache* attribute), 61  
 priority (*SymbolFinder* attribute), 63  
 priority (*WinSwapLayers* attribute), 67  
 priority (*WintelHelper* attribute), 69  
 privileges() (*TOKEN* method), 343  
 Privs (class in *volatility.plugins.windows.privileges*), 514  
 proc (class in *volatility.framework.symbols.mac.extensions*), 298  
 proc.VolTemplateProxy (class in *volatility.framework.symbols.mac.extensions*), 298  
 proc\_filters (*Kevents* attribute), 445  
 process\_dump() (*PsList* class method), 517  
 process\_exceptions() (*CommandLine* method), 28  
 process\_exceptions() (*VolShell* method), 29  
 process\_file\_object() (*DumpFiles* class method), 487  
 process\_index\_array() (*SHARED\_CACHE\_MAP* method), 341  
 process\_types() (*PdbReader* method), 374  
 process\_unsatisfied\_exceptions() (*CommandLine* method), 28  
 process\_unsatisfied\_exceptions() (*VolShell* method), 29  
 ProgressCallback (in module *volatility.framework.constants*), 92  
 protect\_values() (*VadInfo* class method), 529  
 provides (*LinuxKernelIntermedSymbols* attribute), 256  
 provides (*MacKernelIntermedSymbols* attribute), 290  
 provides (*WindowsCrashDump32Layer* attribute), 143  
 provides (*WindowsCrashDump64Layer* attribute), 145  
 Psaux (class in *volatility.plugins.mac.psaux*), 457  
 PsList (class in *volatility.plugins.linux.pslist*), 427  
 PsList (class in *volatility.plugins.mac.pslist*), 459  
 PsList (class in *volatility.plugins.windows.pslist*), 516

pslist\_methods (*PsList* attribute), 461  
PsScan (class in volatility.plugins.windows.psscan), 518  
PsTree (class in volatility.plugins.linux.pstree), 429  
PsTree (class in volatility.plugins.mac.pstree), 462  
PsTree (class in volatility.plugins.windows.pstree), 521  
PYTHONPATH, 25

## Q

QemuStacker (class in volatility.framework.layers.qemu), 181  
QemuSuspendLayer (class in volatility.framework.layers.qemu), 181  
QEVM\_CONFIGURATION (*QemuSuspendLayer* attribute), 181  
QEVM\_EOF (*QemuSuspendLayer* attribute), 181  
QEVM\_SECTION\_END (*QemuSuspendLayer* attribute), 181  
QEVM\_SECTION\_FOOTER (*QemuSuspendLayer* attribute), 181  
QEVM\_SECTION\_FULL (*QemuSuspendLayer* attribute), 181  
QEVM\_SECTION\_PART (*QemuSuspendLayer* attribute), 181  
QEVM\_SECTION\_START (*QemuSuspendLayer* attribute), 181  
QEVM\_SUBSECTION (*QemuSuspendLayer* attribute), 181  
QEVM\_VMDescription (*QemuSuspendLayer* attribute), 181  
qstr (class in volatility.framework.symbols.linux.extensions), 269  
qstr.VolTemplateProxy (class in volatility.framework.symbols.linux.extensions), 269  
queue\_entry (class in volatility.framework.symbols.mac.extensions), 299  
queue\_entry.VolTemplateProxy (class in volatility.framework.symbols.mac.extensions), 300  
QuickTextRenderer (class in volatility.cli.text\_renderer), 42  
quoted\_optional() (in module volatility.cli.text\_renderer), 43

## R

random\_string() (*Volshell* method), 33, 35, 38, 40  
read() (*BufferDataLayer* method), 178  
read() (*DataLayerInterface* method), 117  
read() (*Elf64Layer* method), 148  
read() (*FileHandlerInterface* method), 126  
read() (*FileLayer* method), 180  
read() (*Intel* method), 151  
read() (*Intel32e* method), 153

read() (*IntelPAE* method), 156  
read() (*LayerContainer* method), 119  
read() (*LimeLayer* method), 167  
read() (*LinearlyMappedLayer* method), 170  
read() (*NonLinearlySegmentedLayer* method), 189  
read() (*NullFileHandler* method), 30  
read() (*PdbMSFStream* method), 173  
read() (*PdbMultiStreamFormat* method), 175  
read() (*QemuSuspendLayer* method), 183  
read() (*RegistryHive* method), 186  
read() (*SegmentedLayer* method), 192  
read() (*TranslationLayerInterface* method), 122  
read() (*VmwareLayer* method), 194  
read() (*WindowsCrashDump32Layer* method), 143  
read() (*WindowsCrashDump64Layer* method), 145  
read() (*WindowsIntel* method), 158  
read() (*WindowsIntel32e* method), 160  
read() (*WindowsIntelPAE* method), 162  
read() (*WindowsMixin* method), 165  
readl() (*NullFileHandler* method), 30  
read\_dbi\_stream() (*PdbReader* method), 374  
read\_ipi\_stream() (*PdbReader* method), 374  
read\_necessary\_streams() (*PdbReader* method), 375  
read\_pdb\_info\_stream() (*PdbReader* method), 375  
read\_streams() (*PdbMultiStreamFormat* method), 175  
read\_symbol\_stream() (*PdbReader* method), 375  
read\_tpi\_stream() (*PdbReader* method), 375  
readable() (*FileHandlerInterface* method), 126  
readable() (*NullFileHandler* method), 31  
readall() (*FileHandlerInterface* method), 126  
readall() (*NullFileHandler* method), 31  
readinto() (*FileHandlerInterface* method), 127  
readinto() (*NullFileHandler* method), 31  
readinto1() (*NullFileHandler* method), 31  
readline() (*FileHandlerInterface* method), 127  
readline() (*NullFileHandler* method), 31  
readlines() (*FileHandlerInterface* method), 127  
readlines() (*NullFileHandler* method), 31  
ReadOnlyMapping (class in volatility.framework.interfaces.objects), 124  
real (*Bin* attribute), 241  
real (*BitField* attribute), 200  
real (*Boolean* attribute), 202  
real (*Char* attribute), 209  
real (*Enumeration* attribute), 214  
real (*Float* attribute), 216  
real (*Hex* attribute), 242  
real (*Integer* attribute), 220  
real (*Pointer* attribute), 222  
reconstruct() (*IMAGE\_DOS\_HEADER* method), 349



`record_cached_validations()` (in module `volatility.schemas`), 543  
`recurse_symbol_fulfiller()` (*KernelPDB-Scanner method*), 57  
`reference_count()` (*kobject method*), 262  
`ReferenceTemplate` (class in `volatility.framework.objects.templates`), 235  
`REG_BINARY` (*RegValueTypes attribute*), 369  
`REG_DWORD` (*RegValueTypes attribute*), 369  
`REG_DWORD_BIG_ENDIAN` (*RegValueTypes attribute*), 369  
`REG_EXPAND_SZ` (*RegValueTypes attribute*), 369  
`REG_FULL_RESOURCE_DESCRIPTOR` (*RegValueTypes attribute*), 369  
`REG_LINK` (*RegValueTypes attribute*), 369  
`REG_MULTI_SZ` (*RegValueTypes attribute*), 369  
`REG_NONE` (*RegValueTypes attribute*), 369  
`REG_QWORD` (*RegValueTypes attribute*), 369  
`REG_RESOURCE_LIST` (*RegValueTypes attribute*), 369  
`REG_RESOURCE_REQUIREMENTS_LIST` (*RegValueTypes attribute*), 369  
`REG_SZ` (*RegValueTypes attribute*), 369  
`REG_UNKNOWN` (*RegValueTypes attribute*), 369  
`RegExScanner` (class in `volatility.framework.layers.scanners`), 140  
`register()` (*HelpfulArgParser method*), 44  
`RegistryFormatException`, 184  
`RegistryHive` (class in `volatility.framework.layers.registry`), 184  
`RegistryInvalidIndex`, 187  
`RegKeyFlags` (class in `volatility.framework.symbols.windows.extensions.registry`), 369  
`RegValueTypes` (class in `volatility.framework.symbols.windows.extensions.registry`), 369  
`relative_child_offset()` (*Aggregate-Type.VolTemplateProxy class method*), 196  
`relative_child_offset()` (*Array.VolTemplateProxy class method*), 197  
`relative_child_offset()` (*Bit-Field.VolTemplateProxy class method*), 199  
`relative_child_offset()` (*Boolean.VolTemplateProxy class method*), 201  
`relative_child_offset()` (*Bytes.VolTemplateProxy class method*), 203  
`relative_child_offset()` (*Char.VolTemplateProxy class method*), 208  
`relative_child_offset()` (*ClassType.VolTemplateProxy class method*), 210  
`relative_child_offset()` (*CM\_KEY\_BODY.VolTemplateProxy class method*), 363  
`relative_child_offset()` (*CM\_KEY\_NODE.VolTemplateProxy class method*), 364  
`relative_child_offset()` (*CM\_KEY\_VALUE.VolTemplateProxy class method*), 366  
`relative_child_offset()` (*CMHIVE.VolTemplateProxy class method*), 361  
`relative_child_offset()` (*CONTROL\_AREA.VolTemplateProxy class method*), 316  
`relative_child_offset()` (*dentry.VolTemplateProxy class method*), 257  
`relative_child_offset()` (*DEVICE\_OBJECT.VolTemplateProxy class method*), 317  
`relative_child_offset()` (*DRIVER\_OBJECT.VolTemplateProxy class method*), 319  
`relative_child_offset()` (*elf.VolTemplateProxy class method*), 281  
`relative_child_offset()` (*elf\_phdr.VolTemplateProxy class method*), 282  
`relative_child_offset()` (*elf\_sym.VolTemplateProxy class method*), 284  
`relative_child_offset()` (*Enumeration.VolTemplateProxy class method*), 212  
`relative_child_offset()` (*EPROCESS.VolTemplateProxy class method*), 321  
`relative_child_offset()` (*ETHREAD.VolTemplateProxy class method*), 323  
`relative_child_offset()` (*EX\_FAST\_REF.VolTemplateProxy class method*), 324  
`relative_child_offset()` (*ExecutiveObject.VolTemplateProxy class method*), 352  
`relative_child_offset()` (*FILE\_OBJECT.VolTemplateProxy class method*), 326  
`relative_child_offset()` (*file-glob.VolTemplateProxy class method*), 292  
`relative_child_offset()` (*files\_struct.VolTemplateProxy class method*), 258  
`relative_child_offset()` (*Float.VolTemplateProxy class method*),

215  
relative\_child\_offset ()  
(fs\_struct.VolTemplateProxy class method),  
260  
relative\_child\_offset () (Function.VolTemplateProxy class method), 217  
relative\_child\_offset () (GenericIntelProcess.VolTemplateProxy class method), 252  
relative\_child\_offset () (hist\_entry.VolTemplateProxy class method),  
279  
relative\_child\_offset () (HMAP\_ENTRY.VolTemplateProxy class  
method), 368  
relative\_child\_offset () (ifnet.VolTemplateProxy class method), 294  
relative\_child\_offset () (IM-  
AGE\_DOS\_HEADER.VolTemplateProxy  
class method), 348  
relative\_child\_offset () (IM-  
AGE\_NT\_HEADERS.VolTemplateProxy class  
method), 350  
relative\_child\_offset () (in-  
pcb.VolTemplateProxy class method), 295  
relative\_child\_offset () (Integer.VolTemplateProxy class method), 218  
relative\_child\_offset () (kauth\_scope.VolTemplateProxy class method),  
297  
relative\_child\_offset () (KDDEBUG-  
GER\_DATA64.VolTemplateProxy class  
method), 346  
relative\_child\_offset () (KMU-  
TANT.VolTemplateProxy class method),  
328  
relative\_child\_offset () (kob-  
ject.VolTemplateProxy class method), 261  
relative\_child\_offset () (KSYS-  
TEM\_TIME.VolTemplateProxy class method),  
329  
relative\_child\_offset () (KTHREAD.VolTemplateProxy class method),  
331  
relative\_child\_offset () (LIST\_ENTRY.VolTemplateProxy class  
method), 332  
relative\_child\_offset () (list\_head.VolTemplateProxy class method),  
263  
relative\_child\_offset () (mm\_struct.VolTemplateProxy class method),  
264  
relative\_child\_offset () (MM-  
VAD.VolTemplateProxy class method), 334  
relative\_child\_offset () (MM-  
VAD\_SHORT.VolTemplateProxy class method),  
336  
relative\_child\_offset () (module.VolTemplateProxy class method), 266  
relative\_child\_offset () (mount.VolTemplateProxy class method),  
268  
relative\_child\_offset () (OBJECT\_HEADER.VolTemplateProxy class  
method), 354  
relative\_child\_offset () (OBJECT\_SYMBOLIC\_LINK.VolTemplateProxy  
class method), 338  
relative\_child\_offset () (ObjectInterface.VolTemplateProxy class method), 123  
relative\_child\_offset () (ObjectTemplate  
method), 235  
relative\_child\_offset () (Pointer.VolTemplateProxy class method),  
220  
relative\_child\_offset () (POOL\_HEADER.VolTemplateProxy class  
method), 355  
relative\_child\_offset () (POOL\_HEADER\_VISTA.VolTemplateProxy  
class method), 357  
relative\_child\_offset () (POOL\_TRACKER\_BIG\_PAGES.VolTemplateProxy  
class method), 359  
relative\_child\_offset () (PrimitiveObject.VolTemplateProxy class method), 223  
relative\_child\_offset () (proc.VolTemplateProxy class method), 298  
relative\_child\_offset () (qstr.VolTemplateProxy class method), 269  
relative\_child\_offset () (queue\_entry.VolTemplateProxy class method),  
300  
relative\_child\_offset () (ReferenceTemplate  
method), 235  
relative\_child\_offset () (SER-  
VICE\_HEADER.VolTemplateProxy class  
method), 370  
relative\_child\_offset () (SER-  
VICE\_RECORD.VolTemplateProxy class  
method), 371  
relative\_child\_offset () (SHARED\_CACHE\_MAP.VolTemplateProxy  
class method), 340  
relative\_child\_offset () (sock-  
addr.VolTemplateProxy class method), 302  
relative\_child\_offset () (sock-  
addr\_dl.VolTemplateProxy class method),

303  
relative\_child\_offset() (socket.VolTemplateProxy class method), 304  
relative\_child\_offset() (String.VolTemplateProxy class method), 224  
relative\_child\_offset() (struct\_file.VolTemplateProxy class method), 271  
relative\_child\_offset() (StructType.VolTemplateProxy class method), 230  
relative\_child\_offset() (super\_block.VolTemplateProxy class method), 272  
relative\_child\_offset() (SymbolSpace.UnresolvedTemplate method), 250  
relative\_child\_offset() (sysctl\_oid.VolTemplateProxy class method), 306  
relative\_child\_offset() (task\_struct.VolTemplateProxy class method), 274  
relative\_child\_offset() (Template method), 125  
relative\_child\_offset() (TOKEN.VolTemplateProxy class method), 342  
relative\_child\_offset() (UNICODE\_STRING.VolTemplateProxy class method), 343  
relative\_child\_offset() (UnionType.VolTemplateProxy class method), 231  
relative\_child\_offset() (VACB.VolTemplateProxy class method), 345  
relative\_child\_offset() (vfs\_mount.VolTemplateProxy class method), 275  
relative\_child\_offset() (vm\_area\_struct.VolTemplateProxy class method), 277  
relative\_child\_offset() (vm\_map\_entry.VolTemplateProxy class method), 308  
relative\_child\_offset() (vm\_map\_object.VolTemplateProxy class method), 309  
relative\_child\_offset() (vmode.VolTemplateProxy class method), 311  
relative\_child\_offset() (Void.VolTemplateProxy class method), 233  
remove() (SymbolSpace method), 251  
remove\_requirement() (BooleanRequirement method), 71  
remove\_requirement() (BytesRequirement method), 72  
remove\_requirement() (ChoiceRequirement method), 73  
remove\_requirement() (ClassRequirement method), 102  
remove\_requirement() (ComplexListRequirement method), 75  
remove\_requirement() (ConfigurableRequirementInterface method), 105  
remove\_requirement() (ConstructableRequirementInterface method), 107  
remove\_requirement() (IntRequirement method), 77  
remove\_requirement() (LayerListRequirement method), 78  
remove\_requirement() (ListRequirement method), 80  
remove\_requirement() (MultiRequirement method), 82  
remove\_requirement() (PluginRequirement method), 83  
remove\_requirement() (RequirementInterface method), 110  
remove\_requirement() (SimpleTypeRequirement method), 111  
remove\_requirement() (StringRequirement method), 85  
remove\_requirement() (SymbolTableRequirement method), 86  
remove\_requirement() (TranslationLayerRequirement method), 88  
remove\_requirement() (URIRequirement method), 89  
remove\_requirement() (VersionRequirement method), 91  
render() (CLIRenderer method), 41  
render() (CSVRenderer method), 41  
render() (JsonLinesRenderer method), 41  
render() (JsonRenderer method), 42  
render() (PrettyTextRenderer method), 42  
render() (QuickTextRenderer method), 42  
render() (Renderer method), 130  
render\_treegrid() (Volshell method), 33, 35, 38, 40  
Renderer (class in volatility.framework.interfaces.renderers), 130  
replace() (Bytes method), 206  
replace() (HexBytes method), 244  
replace() (MultiTypeData method), 248  
replace() (String method), 228  
replace\_child() (AggregateType.VolTemplateProxy class method), 196  
replace\_child() (Array.VolTemplateProxy class

method), 197  
replace\_child() (*BitField.VolTemplateProxy* class method), 199  
replace\_child() (*Boolean.VolTemplateProxy* class method), 201  
replace\_child() (*Bytes.VolTemplateProxy* class method), 203  
replace\_child() (*Char.VolTemplateProxy* class method), 208  
replace\_child() (*ClassType.VolTemplateProxy* class method), 210  
replace\_child() (*CM\_KEY\_BODY.VolTemplateProxy* class method), 363  
replace\_child() (*CM\_KEY\_NODE.VolTemplateProxy* class method), 364  
replace\_child() (*CM\_KEY\_VALUE.VolTemplateProxy* class method), 366  
replace\_child() (*CMHIVE.VolTemplateProxy* class method), 361  
replace\_child() (*CONTROL\_AREA.VolTemplateProxy* class method), 316  
replace\_child() (*dentry.VolTemplateProxy* class method), 257  
replace\_child() (*DEVICE\_OBJECT.VolTemplateProxy* class method), 317  
replace\_child() (*DRIVER\_OBJECT.VolTemplateProxy* class method), 319  
replace\_child() (*elf.VolTemplateProxy* class method), 281  
replace\_child() (*elf\_phdr.VolTemplateProxy* class method), 282  
replace\_child() (*elf\_sym.VolTemplateProxy* class method), 284  
replace\_child() (*Enumeration.VolTemplateProxy* class method), 212  
replace\_child() (*EPROCESS.VolTemplateProxy* class method), 321  
replace\_child() (*ETHREAD.VolTemplateProxy* class method), 323  
replace\_child() (*EX\_FAST\_REF.VolTemplateProxy* class method), 324  
replace\_child() (*ExecutiveObject.VolTemplateProxy* class method), 352  
replace\_child() (*FILE\_OBJECT.VolTemplateProxy* class method), 326  
replace\_child() (*fileglob.VolTemplateProxy* class method), 292  
replace\_child() (*files\_struct.VolTemplateProxy* class method), 259  
replace\_child() (*Float.VolTemplateProxy* class method), 215  
replace\_child() (*fs\_struct.VolTemplateProxy* class method), 260  
replace\_child() (*Function.VolTemplateProxy* class method), 217  
replace\_child() (*GenericIntelProcess.VolTemplateProxy* class method), 252  
replace\_child() (*hist\_entry.VolTemplateProxy* class method), 279  
replace\_child() (*HMAP\_ENTRY.VolTemplateProxy* class method), 368  
replace\_child() (*ifnet.VolTemplateProxy* class method), 294  
replace\_child() (*IMAGE\_DOS\_HEADER.VolTemplateProxy* class method), 348  
replace\_child() (*IMAGE\_NT\_HEADERS.VolTemplateProxy* class method), 350  
replace\_child() (*inpcb.VolTemplateProxy* class method), 295  
replace\_child() (*Integer.VolTemplateProxy* class method), 218  
replace\_child() (*kauth\_scope.VolTemplateProxy* class method), 297  
replace\_child() (*KDDEBUGGER\_DATA64.VolTemplateProxy* class method), 346  
replace\_child() (*KMUTANT.VolTemplateProxy* class method), 328  
replace\_child() (*kobject.VolTemplateProxy* class method), 262  
replace\_child() (*KSYSTEM\_TIME.VolTemplateProxy* class method), 329  
replace\_child() (*KTHREAD.VolTemplateProxy* class method), 331  
replace\_child() (*LIST\_ENTRY.VolTemplateProxy* class method), 332  
replace\_child() (*list\_head.VolTemplateProxy* class method), 263  
replace\_child() (*mm\_struct.VolTemplateProxy* class method), 264  
replace\_child() (*MMVAD.VolTemplateProxy* class method), 334  
replace\_child() (*MMVAD\_SHORT.VolTemplateProxy* class method), 336  
replace\_child() (*module.VolTemplateProxy* class method), 266  
replace\_child() (*mount.VolTemplateProxy* class method), 268  
replace\_child() (*OBJECT\_HEADER.VolTemplateProxy* class method), 354  
replace\_child() (*OB-*



[JECT\\_SYMBOLIC\\_LINK.VolTemplateProxy class method](#)), 338  
[replace\\_child\(\)](#) ([ObjectInterface.VolTemplateProxy class method](#)), 123  
[replace\\_child\(\)](#) ([ObjectTemplate method](#)), 235  
[replace\\_child\(\)](#) ([Pointer.VolTemplateProxy class method](#)), 221  
[replace\\_child\(\)](#) ([POOL\\_HEADER.VolTemplateProxy class method](#)), 355  
[replace\\_child\(\)](#) ([POOL\\_HEADER\\_VISTA.VolTemplateProxy class method](#)), 357  
[replace\\_child\(\)](#) ([POOL\\_TRACKER\\_BIG\\_PAGES.VolTemplateProxy class method](#)), 359  
[replace\\_child\(\)](#) ([PrimitiveObject.VolTemplateProxy class method](#)), 223  
[replace\\_child\(\)](#) ([proc.VolTemplateProxy class method](#)), 298  
[replace\\_child\(\)](#) ([qstr.VolTemplateProxy class method](#)), 269  
[replace\\_child\(\)](#) ([queue\\_entry.VolTemplateProxy class method](#)), 300  
[replace\\_child\(\)](#) ([ReferenceTemplate method](#)), 236  
[replace\\_child\(\)](#) ([SERVICE\\_HEADER.VolTemplateProxy class method](#)), 370  
[replace\\_child\(\)](#) ([SERVICE\\_RECORD.VolTemplateProxy class method](#)), 372  
[replace\\_child\(\)](#) ([SHARED\\_CACHE\\_MAP.VolTemplateProxy class method](#)), 340  
[replace\\_child\(\)](#) ([sockaddr.VolTemplateProxy class method](#)), 302  
[replace\\_child\(\)](#) ([sockaddr\\_dl.VolTemplateProxy class method](#)), 303  
[replace\\_child\(\)](#) ([socket.VolTemplateProxy class method](#)), 305  
[replace\\_child\(\)](#) ([String.VolTemplateProxy class method](#)), 224  
[replace\\_child\(\)](#) ([struct\\_file.VolTemplateProxy class method](#)), 271  
[replace\\_child\(\)](#) ([StructType.VolTemplateProxy class method](#)), 230  
[replace\\_child\(\)](#) ([super\\_block.VolTemplateProxy class method](#)), 272  
[replace\\_child\(\)](#) ([SymbolSpace.UnresolvedTemplate method](#)), 250  
[replace\\_child\(\)](#) ([sysctl\\_oid.VolTemplateProxy class method](#)), 306  
[replace\\_child\(\)](#) ([task\\_struct.VolTemplateProxy class method](#)), 274  
[replace\\_child\(\)](#) ([Template method](#)), 125  
[replace\\_child\(\)](#) ([TOKEN.VolTemplateProxy class method](#)), 342  
[replace\\_child\(\)](#) ([UNI](#)

[CODE\\_STRING.VolTemplateProxy class method](#)), 343  
[replace\\_child\(\)](#) ([UnionType.VolTemplateProxy class method](#)), 231  
[replace\\_child\(\)](#) ([VACB.VolTemplateProxy class method](#)), 345  
[replace\\_child\(\)](#) ([vfsmount.VolTemplateProxy class method](#)), 276  
[replace\\_child\(\)](#) ([vm\\_area\\_struct.VolTemplateProxy class method](#)), 277  
[replace\\_child\(\)](#) ([vm\\_map\\_entry.VolTemplateProxy class method](#)), 308  
[replace\\_child\(\)](#) ([vm\\_map\\_object.VolTemplateProxy class method](#)), 310  
[replace\\_child\(\)](#) ([vnode.VolTemplateProxy class method](#)), 311  
[replace\\_child\(\)](#) ([Void.VolTemplateProxy class method](#)), 233  
[replace\\_forward\\_references\(\)](#) ([PdbReader method](#)), 375  
[replace\\_header\\_field\(\)](#) ([IMAGE\\_DOS\\_HEADER method](#)), 349  
[require\\_interface\\_version\(\)](#) ([in module volatility.framework](#)), 44  
[RequirementInterface](#) ([class in volatility.framework.interfaces.configuration](#)), 109  
[requirements\(\)](#) ([BooleanRequirement property](#)), 71  
[requirements\(\)](#) ([BytesRequirement property](#)), 72  
[requirements\(\)](#) ([ChoiceRequirement property](#)), 73  
[requirements\(\)](#) ([ClassRequirement property](#)), 102  
[requirements\(\)](#) ([ComplexListRequirement property](#)), 75  
[requirements\(\)](#) ([ConfigurableRequirementInterface property](#)), 105  
[requirements\(\)](#) ([ConstructableRequirementInterface property](#)), 107  
[requirements\(\)](#) ([IntRequirement property](#)), 77  
[requirements\(\)](#) ([LayerListRequirement property](#)), 78  
[requirements\(\)](#) ([ListRequirement property](#)), 80  
[requirements\(\)](#) ([MultiRequirement property](#)), 82  
[requirements\(\)](#) ([PluginRequirement property](#)), 83  
[requirements\(\)](#) ([RequirementInterface property](#)), 110  
[requirements\(\)](#) ([SimpleTypeRequirement property](#)), 111  
[requirements\(\)](#) ([StringRequirement property](#)), 85  
[requirements\(\)](#) ([SymbolTableRequirement property](#)), 86  
[requirements\(\)](#) ([TranslationLayerRequirement property](#)), 88  
[requirements\(\)](#) ([URIRequirement property](#)), 89  
[requirements\(\)](#) ([VersionRequirement property](#)), 91  
[reset\(\)](#) ([PdbReader method](#)), 375

ResourceAccessor (class in volatility.framework.layers.resources), 187  
retrieve\_pdb() (PdbRetreiver method), 375  
rfind() (Bytes method), 206  
rfind() (HexBytes method), 244  
rfind() (MultiTypeData method), 248  
rfind() (String method), 228  
rindex() (Bytes method), 206  
rindex() (HexBytes method), 244  
rindex() (MultiTypeData method), 248  
rindex() (String method), 228  
rjust() (Bytes method), 206  
rjust() (HexBytes method), 245  
rjust() (MultiTypeData method), 248  
rjust() (String method), 228  
root\_cell\_offset() (RegistryHive property), 186  
round() (in module volatility.framework.renderers.conversion), 240  
row\_count() (TreeGrid property), 238  
RowStructureConstructor() (in module volatility.framework.renderers), 237  
rpartition() (Bytes method), 206  
rpartition() (HexBytes method), 245  
rpartition() (MultiTypeData method), 248  
rpartition() (String method), 228  
rsplit() (Bytes method), 206  
rsplit() (HexBytes method), 245  
rsplit() (MultiTypeData method), 248  
rsplit() (String method), 228  
rstrip() (Bytes method), 207  
rstrip() (HexBytes method), 245  
rstrip() (MultiTypeData method), 248  
rstrip() (String method), 229  
run() (Banners method), 534  
run() (Bash method), 410, 434  
run() (BigPools method), 478  
run() (Check\_afinfo method), 411  
run() (Check\_creds method), 413  
run() (Check\_idt method), 414  
run() (Check\_modules method), 416  
run() (Check\_syscall method), 417, 435  
run() (Check\_sysctl method), 437  
run() (Check\_trap\_table method), 438  
run() (CmdLine method), 480  
run() (CommandLine method), 28  
run() (ConfigWriter method), 536  
run() (DllList method), 482  
run() (DriverIrp method), 483  
run() (DriverScan method), 485  
run() (DumpFiles method), 487  
run() (Elfs method), 419  
run() (Envvars method), 489  
run() (FileScan method), 490  
run() (FrameworkInfo method), 537  
run() (GetServiceSIDs method), 492  
run() (GetSIDs method), 494  
run() (Handles method), 496  
run() (HiveList method), 471  
run() (HiveScan method), 473  
run() (Ifconfig method), 440  
run() (in module volatility.framework.automagic), 45  
run() (Info method), 498  
run() (IsfInfo method), 539  
run() (Kauth\_listeners method), 441  
run() (Kauth\_scopes method), 443  
run() (Kevents method), 445  
run() (Keyboard\_notifiers method), 420  
run() (LayerWriter method), 541  
run() (List\_Files method), 447  
run() (Lsmmod method), 422, 448  
run() (Lsof method), 424, 450  
run() (Malfind method), 425, 451, 500  
run() (Maps method), 427, 456  
run() (Memmap method), 501  
run() (ModScan method), 504  
run() (Modules method), 506  
run() (Mount method), 453  
run() (MutantScan method), 508  
run() (NetScan method), 510  
run() (Netstat method), 455  
run() (PluginInterface method), 128  
run() (PoolScanner method), 513  
run() (PrintKey method), 475  
run() (Privs method), 515  
run() (Psaux method), 458  
run() (PsList method), 429, 461, 518  
run() (PsScan method), 519  
run() (PsTree method), 431, 463, 521  
run() (Socket\_filters method), 464  
run() (SSDT method), 523  
run() (Strings method), 525  
run() (SymlinkScan method), 527  
run() (Timeliner method), 543  
run() (Timers method), 466  
run() (Trustedbsd method), 467  
run() (tty\_check method), 432  
run() (UserAssist method), 477  
run() (VadInfo method), 529  
run() (VerInfo method), 531  
run() (VFSevents method), 469  
run() (VirtMap method), 533  
run() (VolShell method), 30  
run() (Volshell method), 33, 35, 38, 40

## S

sanitize\_name() (TreeGrid static method), 131, 238  
save\_banners() (LinuxBannerCache class method), 48

- save\_banners() (*MacBannerCache* class method), 52
- save\_banners() (*SymbolBannerCache* class method), 61
- save\_vacb() (*SHARED\_CACHE\_MAP* method), 341
- scan() (*BufferDataLayer* method), 178
- scan() (*DataLayerInterface* method), 118
- scan() (*Elf64Layer* method), 148
- scan() (*FileLayer* method), 180
- scan() (*Intel* method), 151
- scan() (*Intel32e* method), 153
- scan() (*IntelPAE* method), 156
- scan() (*LimeLayer* method), 167
- scan() (*LinearlyMappedLayer* method), 170
- scan() (*NetScan* class method), 510
- scan() (*NonLinearlySegmentedLayer* method), 190
- scan() (*PdbMSFStream* method), 173
- scan() (*PdbMultiStreamFormat* method), 175
- scan() (*QemuSuspendLayer* method), 183
- scan() (*RegistryHive* method), 186
- scan() (*SegmentedLayer* method), 192
- scan() (*TranslationLayerInterface* method), 122
- scan() (*VmwareLayer* method), 194
- scan() (*WindowsCrashDump32Layer* method), 143
- scan() (*WindowsCrashDump64Layer* method), 145
- scan() (*WindowsIntel* method), 158
- scan() (*WindowsIntel32e* method), 160
- scan() (*WindowsIntelPAE* method), 162
- scan() (*WindowsMixin* method), 165
- scan\_drivers() (*DriverScan* class method), 485
- scan\_files() (*FileScan* class method), 491
- scan\_hives() (*HiveScan* class method), 473
- scan\_modules() (*ModScan* class method), 504
- scan\_mutants() (*MutantScan* class method), 508
- scan\_processes() (*PsScan* class method), 520
- scan\_symlinks() (*SymlinkScan* class method), 527
- scannable\_sections() (*VirtMap* class method), 533
- ScannerInterface (class in *volatility.framework.interfaces.layers*), 119
- search() (*MultiRegexp* method), 141
- second\_pass() (*DtbSelfRef32bit* method), 64
- second\_pass() (*DtbSelfRef64bit* method), 64
- second\_pass() (*DtbSelfReferential* method), 64
- second\_pass() (*DtbTest* method), 65
- second\_pass() (*DtbTest32bit* method), 65
- second\_pass() (*DtbTest64bit* method), 65
- second\_pass() (*DtbTestPae* method), 66
- section\_strtab() (module property), 267
- section\_syntab() (module property), 267
- seek() (*FileHandlerInterface* method), 127
- seek() (*NullFileHandler* method), 31
- seekable() (*FileHandlerInterface* method), 127
- seekable() (*NullFileHandler* method), 31
- SEGMENT\_FLAG\_COMPRESS (*QemuSuspendLayer* attribute), 181
- SEGMENT\_FLAG\_CONTINUE (*QemuSuspendLayer* attribute), 182
- SEGMENT\_FLAG\_EOS (*QemuSuspendLayer* attribute), 182
- SEGMENT\_FLAG\_HOOK (*QemuSuspendLayer* attribute), 182
- SEGMENT\_FLAG\_MEM\_SIZE (*QemuSuspendLayer* attribute), 182
- SEGMENT\_FLAG\_PAGE (*QemuSuspendLayer* attribute), 182
- SEGMENT\_FLAG\_XBZRLE (*QemuSuspendLayer* attribute), 182
- SegmentedLayer (class in *volatility.framework.layers.segmented*), 190
- separator() (*HierarchicalDict* property), 108
- SERVICE\_HEADER (class in *volatility.framework.symbols.windows.extensions.services*), 370
- SERVICE\_HEADER.VolTemplateProxy (class in *volatility.framework.symbols.windows.extensions.services*), 370
- SERVICE\_RECORD (class in *volatility.framework.symbols.windows.extensions.services*), 371
- SERVICE\_RECORD.VolTemplateProxy (class in *volatility.framework.symbols.windows.extensions.services*), 371
- set\_defaults() (*HelpfulArgParser* method), 44
- set\_kernel\_virtual\_offset() (*KernelPDB-Scanner* method), 57
- set\_open\_method() (*Banners* method), 534
- set\_open\_method() (*Bash* method), 410, 434
- set\_open\_method() (*BigPools* method), 478
- set\_open\_method() (*Check\_afinfo* method), 411
- set\_open\_method() (*Check\_creds* method), 413
- set\_open\_method() (*Check\_idt* method), 414
- set\_open\_method() (*Check\_modules* method), 416
- set\_open\_method() (*Check\_syscall* method), 417, 436
- set\_open\_method() (*Check\_sysctl* method), 437
- set\_open\_method() (*Check\_trap\_table* method), 439
- set\_open\_method() (*CmdLine* method), 480
- set\_open\_method() (*ConfigWriter* method), 536
- set\_open\_method() (*DllList* method), 482
- set\_open\_method() (*DriverIrp* method), 484
- set\_open\_method() (*DriverScan* method), 485
- set\_open\_method() (*DumpFiles* method), 487
- set\_open\_method() (*Elfs* method), 419
- set\_open\_method() (*Envvars* method), 489

`set_open_method()` (*FileScan method*), 491  
`set_open_method()` (*FrameworkInfo method*), 537  
`set_open_method()` (*GetServiceSIDs method*), 492  
`set_open_method()` (*GetSIDs method*), 494  
`set_open_method()` (*Handles method*), 496  
`set_open_method()` (*HiveList method*), 472  
`set_open_method()` (*HiveScan method*), 473  
`set_open_method()` (*Ifconfig method*), 440  
`set_open_method()` (*Info method*), 498  
`set_open_method()` (*IsfInfo method*), 539  
`set_open_method()` (*Kauth\_listeners method*), 442  
`set_open_method()` (*Kauth\_scopes method*), 443  
`set_open_method()` (*Kevents method*), 445  
`set_open_method()` (*Keyboard\_notifiers method*), 421  
`set_open_method()` (*LayerWriter method*), 541  
`set_open_method()` (*List\_Files method*), 447  
`set_open_method()` (*Lsmode method*), 422, 448  
`set_open_method()` (*Lsof method*), 424, 450  
`set_open_method()` (*Malfind method*), 425, 451, 500  
`set_open_method()` (*Maps method*), 427, 456  
`set_open_method()` (*Memmap method*), 502  
`set_open_method()` (*ModScan method*), 504  
`set_open_method()` (*Modules method*), 506  
`set_open_method()` (*Mount method*), 453  
`set_open_method()` (*MutantScan method*), 508  
`set_open_method()` (*NetScan method*), 511  
`set_open_method()` (*Netstat method*), 455  
`set_open_method()` (*PluginInterface method*), 129  
`set_open_method()` (*PoolScanner method*), 514  
`set_open_method()` (*PrintKey method*), 475  
`set_open_method()` (*Privs method*), 515  
`set_open_method()` (*Psaux method*), 458  
`set_open_method()` (*PsList method*), 429, 461, 518  
`set_open_method()` (*PsScan method*), 520  
`set_open_method()` (*PsTree method*), 431, 463, 522  
`set_open_method()` (*Socket\_filters method*), 464  
`set_open_method()` (*SSDT method*), 523  
`set_open_method()` (*Strings method*), 525  
`set_open_method()` (*SymlinkScan method*), 527  
`set_open_method()` (*Timeliner method*), 543  
`set_open_method()` (*Timers method*), 466  
`set_open_method()` (*Trustedbsd method*), 467  
`set_open_method()` (*tty\_check method*), 432  
`set_open_method()` (*UserAssist method*), 477  
`set_open_method()` (*VadInfo method*), 529  
`set_open_method()` (*VerInfo method*), 531  
`set_open_method()` (*VFSevents method*), 469  
`set_open_method()` (*VirtMap method*), 533  
`set_open_method()` (*Volshell method*), 33, 36, 38, 40  
`set_type_class()` (*BaseSymbolTableInterface method*), 133  
`set_type_class()` (*BashIntermedSymbols method*), 287  
`set_type_class()` (*IntermediateSymbolTable method*), 383  
`set_type_class()` (*ISFormatTable method*), 379  
`set_type_class()` (*LinuxKernelIntermedSymbols method*), 256  
`set_type_class()` (*MacKernelIntermedSymbols method*), 291  
`set_type_class()` (*NativeTable method*), 406  
`set_type_class()` (*NativeTableInterface method*), 135  
`set_type_class()` (*SymbolTableInterface method*), 139  
`set_type_class()` (*Version1Format method*), 385  
`set_type_class()` (*Version2Format method*), 388  
`set_type_class()` (*Version3Format method*), 390  
`set_type_class()` (*Version4Format method*), 393  
`set_type_class()` (*Version5Format method*), 396  
`set_type_class()` (*Version6Format method*), 398  
`set_type_class()` (*Version7Format method*), 401  
`set_type_class()` (*Version8Format method*), 403  
`set_type_class()` (*WindowsKernelIntermedSymbols method*), 314  
`setter()` (*classproperty method*), 27  
`setup_logging()` (*CommandLine class method*), 28  
`setup_logging()` (*VolShell class method*), 30  
`SHARED_CACHE_MAP` (*class in volatility.framework.symbols.windows.extensions*), 339  
`SHARED_CACHE_MAP.VolTemplateProxy` (*class in volatility.framework.symbols.windows.extensions*), 339  
`SIGNATURE` (*WindowsCrashDump32Layer attribute*), 141  
`SIGNATURE` (*WindowsCrashDump64Layer attribute*), 144  
`signed()` (*DataFormatInfo property*), 211  
`SimpleTypeRequirement` (*class in volatility.framework.interfaces.configuration*), 110  
`size()` (*AggregateType.VolTemplateProxy class method*), 196  
`size()` (*Array.VolTemplateProxy class method*), 197  
`size()` (*BitField.VolTemplateProxy class method*), 199  
`size()` (*Boolean.VolTemplateProxy class method*), 201  
`size()` (*Bytes.VolTemplateProxy class method*), 203  
`size()` (*Char.VolTemplateProxy class method*), 208  
`size()` (*ClassType.VolTemplateProxy class method*), 210  
`size()` (*CM\_KEY\_BODY.VolTemplateProxy class method*), 363  
`size()` (*CM\_KEY\_NODE.VolTemplateProxy class method*), 364



[size\(\) \(CM\\_KEY\\_VALUE.VolTemplateProxy class method\), 366](#)  
[size\(\) \(CMHIVE.VolTemplateProxy class method\), 361](#)  
[size\(\) \(CONTROL\\_AREA.VolTemplateProxy class method\), 316](#)  
[size\(\) \(dentry.VolTemplateProxy class method\), 257](#)  
[size\(\) \(DEVICE\\_OBJECT.VolTemplateProxy class method\), 317](#)  
[size\(\) \(DRIVER\\_OBJECT.VolTemplateProxy class method\), 319](#)  
[size\(\) \(elf.VolTemplateProxy class method\), 281](#)  
[size\(\) \(elf\\_phdr.VolTemplateProxy class method\), 282](#)  
[size\(\) \(elf\\_sym.VolTemplateProxy class method\), 284](#)  
[size\(\) \(Enumeration.VolTemplateProxy class method\), 212](#)  
[size\(\) \(EPROCESS.VolTemplateProxy class method\), 321](#)  
[size\(\) \(ETHREAD.VolTemplateProxy class method\), 323](#)  
[size\(\) \(EX\\_FAST\\_REF.VolTemplateProxy class method\), 325](#)  
[size\(\) \(ExecutiveObject.VolTemplateProxy class method\), 352](#)  
[size\(\) \(FILE\\_OBJECT.VolTemplateProxy class method\), 326](#)  
[size\(\) \(fileglob.VolTemplateProxy class method\), 292](#)  
[size\(\) \(files\\_struct.VolTemplateProxy class method\), 259](#)  
[size\(\) \(Float.VolTemplateProxy class method\), 215](#)  
[size\(\) \(fs\\_struct.VolTemplateProxy class method\), 260](#)  
[size\(\) \(Function.VolTemplateProxy class method\), 217](#)  
[size\(\) \(GenericIntelProcess.VolTemplateProxy class method\), 252](#)  
[size\(\) \(hist\\_entry.VolTemplateProxy class method\), 279](#)  
[size\(\) \(HMAP\\_ENTRY.VolTemplateProxy class method\), 368](#)  
[size\(\) \(ifnet.VolTemplateProxy class method\), 294](#)  
[size\(\) \(IMAGE\\_DOS\\_HEADER.VolTemplateProxy class method\), 348](#)  
[size\(\) \(IMAGE\\_NT\\_HEADERS.VolTemplateProxy class method\), 350](#)  
[size\(\) \(inpcb.VolTemplateProxy class method\), 295](#)  
[size\(\) \(Integer.VolTemplateProxy class method\), 218](#)  
[size\(\) \(kauth\\_scope.VolTemplateProxy class method\), 297](#)  
[size\(\) \(KDDEBUGGER\\_DATA64.VolTemplateProxy class method\), 347](#)  
[size\(\) \(KMUTANT.VolTemplateProxy class method\), 328](#)  
[size\(\) \(kobject.VolTemplateProxy class method\), 262](#)  
[size\(\) \(KSYSTEM\\_TIME.VolTemplateProxy class method\), 329](#)  
[size\(\) \(KTHREAD.VolTemplateProxy class method\), 331](#)  
[size\(\) \(LIST\\_ENTRY.VolTemplateProxy class method\), 332](#)  
[size\(\) \(list\\_head.VolTemplateProxy class method\), 263](#)  
[size\(\) \(mm\\_struct.VolTemplateProxy class method\), 265](#)  
[size\(\) \(MMVAD.VolTemplateProxy class method\), 334](#)  
[size\(\) \(MMVAD\\_SHORT.VolTemplateProxy class method\), 336](#)  
[size\(\) \(module.VolTemplateProxy class method\), 266](#)  
[size\(\) \(mount.VolTemplateProxy class method\), 268](#)  
[size\(\) \(OBJECT\\_HEADER.VolTemplateProxy class method\), 354](#)  
[size\(\) \(OBJECT\\_SYMBOLIC\\_LINK.VolTemplateProxy class method\), 338](#)  
[size\(\) \(ObjectInterface.VolTemplateProxy class method\), 124](#)  
[size\(\) \(ObjectTemplate property\), 235](#)  
[size\(\) \(Pointer.VolTemplateProxy class method\), 221](#)  
[size\(\) \(POOL\\_HEADER.VolTemplateProxy class method\), 355](#)  
[size\(\) \(POOL\\_HEADER\\_VISTA.VolTemplateProxy class method\), 357](#)  
[size\(\) \(POOL\\_TRACKER\\_BIG\\_PAGES.VolTemplateProxy class method\), 359](#)  
[size\(\) \(PrimitiveObject.VolTemplateProxy class method\), 223](#)  
[size\(\) \(proc.VolTemplateProxy class method\), 298](#)  
[size\(\) \(qstr.VolTemplateProxy class method\), 269](#)  
[size\(\) \(queue\\_entry.VolTemplateProxy class method\), 300](#)  
[size\(\) \(ReferenceTemplate property\), 236](#)  
[size\(\) \(SERVICE\\_HEADER.VolTemplateProxy class method\), 370](#)  
[size\(\) \(SERVICE\\_RECORD.VolTemplateProxy class method\), 372](#)  
[size\(\) \(SHARED\\_CACHE\\_MAP.VolTemplateProxy class method\), 340](#)  
[size\(\) \(SizedModule property\), 98](#)  
[size\(\) \(sockaddr.VolTemplateProxy class method\), 302](#)  
[size\(\) \(sockaddr\\_dl.VolTemplateProxy class method\), 303](#)  
[size\(\) \(socket.VolTemplateProxy class method\), 305](#)  
[size\(\) \(String.VolTemplateProxy class method\), 225](#)  
[size\(\) \(struct\\_file.VolTemplateProxy class method\), 271](#)  
[size\(\) \(StructType.VolTemplateProxy class method\), 230](#)  
[size\(\) \(super\\_block.VolTemplateProxy class method\), 272](#)  
[size\(\) \(SymbolSpace.UnresolvedTemplate property\), 250](#)  
[size\(\) \(sysctl\\_oid.VolTemplateProxy class method\),](#)

306  
size() (*task\_struct.VolTemplateProxy* class method), 274  
size() (*Template* property), 125  
size() (*TOKEN.VolTemplateProxy* class method), 342  
size() (*UNICODE\_STRING.VolTemplateProxy* class method), 343  
size() (*UnionType.VolTemplateProxy* class method), 231  
size() (*VACB.VolTemplateProxy* class method), 345  
size() (*vfsmount.VolTemplateProxy* class method), 276  
size() (*vm\_area\_struct.VolTemplateProxy* class method), 277  
size() (*vm\_map\_entry.VolTemplateProxy* class method), 308  
size() (*vm\_map\_object.VolTemplateProxy* class method), 310  
size() (*vnnode.VolTemplateProxy* class method), 311  
size() (*Void.VolTemplateProxy* class method), 233  
SizedModule (class in *volatility.framework.contexts*), 96  
sockaddr (class in *volatility.framework.symbols.mac.extensions*), 301  
sockaddr.VolTemplateProxy (class in *volatility.framework.symbols.mac.extensions*), 301  
sockaddr\_dl (class in *volatility.framework.symbols.mac.extensions*), 303  
sockaddr\_dl() (*ifnet* method), 295  
sockaddr\_dl.VolTemplateProxy (class in *volatility.framework.symbols.mac.extensions*), 303  
socket (class in *volatility.framework.symbols.mac.extensions*), 304  
socket.VolTemplateProxy (class in *volatility.framework.symbols.mac.extensions*), 304  
Socket\_filters (class in *volatility.plugins.mac.socket\_filters*), 463  
splice() (*HierarchicalDict* method), 108  
split() (*Bytes* method), 207  
split() (*HexBytes* method), 245  
split() (*MultiTypeData* method), 248  
split() (*String* method), 229  
splitlines() (*Bytes* method), 207  
splitlines() (*HexBytes* method), 245  
splitlines() (*MultiTypeData* method), 248  
splitlines() (*String* method), 229  
SSDT (class in *volatility.plugins.windows.ssd*), 522  
stack() (*Elf64Stacker* class method), 149  
stack() (*LayerStacker* method), 59  
stack() (*LimeStacker* class method), 168  
stack() (*LinuxIntelStacker* class method), 49  
stack() (*MacIntelStacker* class method), 52  
stack() (*QemuStacker* class method), 181  
stack() (*StackerLayerInterface* class method), 101  
stack() (*VmwareStacker* class method), 195  
stack() (*WindowsCrashDumpStacker* class method), 146  
stack() (*WindowsIntelStacker* class method), 68  
stack\_layer() (*LayerStacker* class method), 59  
stack\_order (*Elf64Stacker* attribute), 149  
stack\_order (*LimeStacker* attribute), 168  
stack\_order (*LinuxIntelStacker* attribute), 49  
stack\_order (*MacIntelStacker* attribute), 52  
stack\_order (*QemuStacker* attribute), 181  
stack\_order (*StackerLayerInterface* attribute), 101  
stack\_order (*VmwareStacker* attribute), 195  
stack\_order (*WindowsCrashDumpStacker* attribute), 147  
stack\_order (*WindowsIntelStacker* attribute), 68  
stacker\_slow\_warning() (*Elf64Stacker* class method), 149  
stacker\_slow\_warning() (*LimeStacker* class method), 168  
stacker\_slow\_warning() (*LinuxIntelStacker* class method), 49  
stacker\_slow\_warning() (*MacIntelStacker* class method), 52  
stacker\_slow\_warning() (*QemuStacker* class method), 181  
stacker\_slow\_warning() (*StackerLayerInterface* class method), 101  
stacker\_slow\_warning() (*VmwareStacker* class method), 195  
stacker\_slow\_warning() (*WindowsCrashDumpStacker* class method), 147  
stacker\_slow\_warning() (*WindowsIntelStacker* class method), 68  
StackerLayerInterface (class in *volatility.framework.interfaces.automagic*), 100  
startswith() (*Bytes* method), 207  
startswith() (*HexBytes* method), 245  
startswith() (*MultiTypeData* method), 249  
startswith() (*String* method), 229  
String (class in *volatility.framework.objects*), 224  
String() (*UNICODE\_STRING* property), 343  
String.VolTemplateProxy (class in *volatility.framework.objects*), 224  
StringRequirement (class in *volatility.framework.configuration.requirements*), 84  
Strings (class in *volatility.plugins.windows.strings*), 524  
strings\_pattern (*Strings* attribute), 525  
strip() (*Bytes* method), 207  
strip() (*HexBytes* method), 245  
strip() (*MultiTypeData* method), 249  
strip() (*String* method), 229  
struct\_file (class in *volatil-*

[ity.framework.symbols.linux.extensions](#)),  
[270](#)  
[struct\\_file.VolTemplateProxy](#) (class in [volatility.framework.symbols.linux.extensions](#)),  
[271](#)  
[StructType](#) (class in [volatility.framework.objects](#)),  
[229](#)  
[StructType.VolTemplateProxy](#) (class in [volatility.framework.objects](#)), [230](#)  
[structure](#) ([Intel](#) attribute), [151](#)  
[structure](#) ([Intel32e](#) attribute), [154](#)  
[structure](#) ([IntelPAE](#) attribute), [156](#)  
[structure](#) ([WindowsIntel](#) attribute), [158](#)  
[structure](#) ([WindowsIntel32e](#) attribute), [160](#)  
[structure](#) ([WindowsIntelPAE](#) attribute), [163](#)  
[structure](#) ([WindowsMixin](#) attribute), [165](#)  
[structured\\_output](#) ([CLIRenderer](#) attribute), [41](#)  
[structured\\_output](#) ([CSVRenderer](#) attribute), [41](#)  
[structured\\_output](#) ([JsonLinesRenderer](#) attribute),  
[41](#)  
[structured\\_output](#) ([JsonRenderer](#) attribute), [42](#)  
[structured\\_output](#) ([PrettyTextRenderer](#) attribute),  
[42](#)  
[structured\\_output](#) ([QuickTextRenderer](#) attribute),  
[42](#)  
[super\\_block](#) (class in [volatility.framework.symbols.linux.extensions](#)),  
[272](#)  
[super\\_block.VolTemplateProxy](#) (class in [volatility.framework.symbols.linux.extensions](#)),  
[272](#)  
[supported\\_dumptypes](#) ([WindowsCrash-Dump32Layer](#) attribute), [143](#)  
[supported\\_dumptypes](#) ([WindowsCrash-Dump64Layer](#) attribute), [146](#)  
[swapcase](#) () ([Bytes](#) method), [207](#)  
[swapcase](#) () ([HexBytes](#) method), [245](#)  
[swapcase](#) () ([MultiTypeData](#) method), [249](#)  
[swapcase](#) () ([String](#) method), [229](#)  
[SwappedInvalidAddressException](#), [407](#)  
[SYMBOL](#) ([SymbolType](#) attribute), [251](#)  
[SYMBOL\\_BASEPATHS](#) (in module [volatility.framework.constants](#)), [93](#)  
[symbol\\_class](#) ([LinuxSymbolFinder](#) attribute), [50](#)  
[symbol\\_class](#) ([MacSymbolFinder](#) attribute), [54](#)  
[symbol\\_class](#) ([SymbolFinder](#) attribute), [63](#)  
[symbol\\_name](#) ([LinuxBannerCache](#) attribute), [48](#)  
[symbol\\_name](#) ([MacBannerCache](#) attribute), [52](#)  
[symbol\\_name](#) ([SymbolBannerCache](#) attribute), [62](#)  
[symbol\\_space](#) () ([Context](#) property), [94](#)  
[symbol\\_space](#) () ([ContextInterface](#) property), [114](#)  
[symbol\\_table\\_from\\_offset](#) () ([PDBUtility](#) class  
method), [376](#)  
[symbol\\_table\\_is\\_64bit](#) () (in module [volatility.framework.symbols](#)), [251](#)  
[SymbolBannerCache](#) (class in [volatility.framework.automagic.symbol\\_cache](#)),  
[60](#)  
[SymbolError](#), [407](#)  
[SymbolFinder](#) (class in [volatility.framework.automagic.symbol\\_finder](#)),  
[62](#)  
[SymbolInterface](#) (class in [volatility.framework.interfaces.symbols](#)), [135](#)  
[symbols](#) () ([BaseSymbolTableInterface](#) property), [134](#)  
[symbols](#) () ([BashIntermedSymbols](#) property), [288](#)  
[symbols](#) () ([IntermediateSymbolTable](#) property), [383](#)  
[symbols](#) () ([ISFormatTable](#) property), [379](#)  
[symbols](#) () ([LinuxKernelIntermedSymbols](#) property),  
[256](#)  
[symbols](#) () ([MacKernelIntermedSymbols](#) property),  
[291](#)  
[symbols](#) () ([NativeTable](#) property), [406](#)  
[symbols](#) () ([NativeTableInterface](#) property), [135](#)  
[symbols](#) () ([SymbolTableInterface](#) property), [139](#)  
[symbols](#) () ([Version1Format](#) property), [385](#)  
[symbols](#) () ([Version2Format](#) property), [388](#)  
[symbols](#) () ([Version3Format](#) property), [390](#)  
[symbols](#) () ([Version4Format](#) property), [393](#)  
[symbols](#) () ([Version5Format](#) property), [396](#)  
[symbols](#) () ([Version6Format](#) property), [398](#)  
[symbols](#) () ([Version7Format](#) property), [401](#)  
[symbols](#) () ([Version8Format](#) property), [404](#)  
[symbols](#) () ([WindowsKernelIntermedSymbols](#) property), [315](#)  
[SymbolSpace](#) (class in [volatility.framework.symbols](#)),  
[249](#)  
[SymbolSpace.UnresolvedTemplate](#) (class in [volatility.framework.symbols](#)), [249](#)  
[SymbolSpaceError](#), [408](#)  
[SymbolSpaceInterface](#) (class in [volatility.framework.interfaces.symbols](#)), [136](#)  
[SymbolTableInterface](#) (class in [volatility.framework.interfaces.symbols](#)), [137](#)  
[SymbolTableRequirement](#) (class in [volatility.framework.configuration.requirements](#)),  
[85](#)  
[SymbolType](#) (class in [volatility.framework.symbols](#)),  
[251](#)  
[SymlinkScan](#) (class in [volatility.plugins.windows.symlinkscan](#)), [526](#)  
[sysctl\\_oid](#) (class in [volatility.framework.symbols.mac.extensions](#)), [306](#)  
[sysctl\\_oid.VolTemplateProxy](#) (class in [volatility.framework.symbols.mac.extensions](#)), [306](#)

## T

- `task_struct` (class in `volatility.framework.symbols.linux.extensions`), 273
- `task_struct.VolTemplateProxy` (class in `volatility.framework.symbols.linux.extensions`), 274
- `tell()` (*FileHandlerInterface* method), 127
- `tell()` (*NullFileHandler* method), 31
- `Template` (class in `volatility.framework.interfaces.objects`), 125
- `tests` (*PageMapScanner* attribute), 66
- `tests` (*WintelHelper* attribute), 69
- `thread_safe` (*BytesScanner* attribute), 140
- `thread_safe` (*MultiStringScanner* attribute), 140
- `thread_safe` (*PageMapScanner* attribute), 66
- `thread_safe` (*PdbSignatureScanner* attribute), 376
- `thread_safe` (*PoolHeaderScanner* attribute), 511
- `thread_safe` (*RegExScanner* attribute), 140
- `thread_safe` (*ScannerInterface* attribute), 120
- `Threading` (*Parallelism* attribute), 92
- `Timeliner` (class in `volatility.plugins.timeliner`), 542
- `TimeLinerInterface` (class in `volatility.plugins.timeliner`), 542
- `TimeLinerType` (class in `volatility.plugins.timeliner`), 542
- `timer_filters` (*Kevents* attribute), 445
- `Timers` (class in `volatility.plugins.mac.timers`), 465
- `title()` (*Bytes* method), 207
- `title()` (*HexBytes* method), 245
- `title()` (*MultiTypeData* method), 249
- `title()` (*String* method), 229
- `to_bytes()` (*Bin* method), 241
- `to_bytes()` (*BitField* method), 200
- `to_bytes()` (*Boolean* method), 202
- `to_bytes()` (*Char* method), 209
- `to_bytes()` (*Enumeration* method), 214
- `to_bytes()` (*Hex* method), 242
- `to_bytes()` (*Integer* method), 220
- `to_bytes()` (*Pointer* method), 222
- `to_list()` (*LIST\_ENTRY* method), 333
- `to_list()` (*list\_head* method), 264
- `TOKEN` (class in `volatility.framework.symbols.windows.extensions`), 341
- `TOKEN.VolTemplateProxy` (class in `volatility.framework.symbols.windows.extensions`), 341
- `translate()` (*Bytes* method), 207
- `translate()` (*Elf64Layer* method), 149
- `translate()` (*HexBytes* method), 245
- `translate()` (*Intel* method), 152
- `translate()` (*Intel32e* method), 154
- `translate()` (*IntelPAE* method), 156
- `translate()` (*LimeLayer* method), 168
- `translate()` (*LinearlyMappedLayer* method), 171
- `translate()` (*MultiTypeData* method), 249
- `translate()` (*PdbMSFStream* method), 173
- `translate()` (*PdbMultiStreamFormat* method), 176
- `translate()` (*RegistryHive* method), 186
- `translate()` (*SegmentedLayer* method), 192
- `translate()` (*String* method), 229
- `translate()` (*VmwareLayer* method), 195
- `translate()` (*WindowsCrashDump32Layer* method), 143
- `translate()` (*WindowsCrashDump64Layer* method), 146
- `translate()` (*WindowsIntel* method), 158
- `translate()` (*WindowsIntel32e* method), 160
- `translate()` (*WindowsIntelPAE* method), 163
- `translate()` (*WindowsMixin* method), 165
- `TranslationLayerInterface` (class in `volatility.framework.interfaces.layers`), 120
- `TranslationLayerRequirement` (class in `volatility.framework.configuration.requirements`), 87
- `traverse()` (*MMVAD* method), 335
- `traverse()` (*MMVAD\_SHORT* method), 337
- `traverse()` (*SERVICE\_RECORD* method), 373
- `TreeGrid` (class in `volatility.framework.interfaces.renderers`), 130
- `TreeGrid` (class in `volatility.framework.renderers`), 237
- `TreeNode` (class in `volatility.framework.interfaces.renderers`), 132
- `TreeNode` (class in `volatility.framework.renderers`), 239
- `truncate()` (*FileHandlerInterface* method), 127
- `truncate()` (*NullFileHandler* method), 31
- `Trustedbsd` (class in `volatility.plugins.mac.trustedbsd`), 466
- `tty_check` (class in `volatility.plugins.linux.tty_check`), 431
- `TYPE` (*SymbolType* attribute), 251
- `type()` (*Column* property), 129
- `type()` (*SymbolInterface* property), 136
- `type_name()` (*SymbolInterface* property), 136
- `type_prefix()` (*elf\_phdr* property), 283
- `types()` (*BaseSymbolTableInterface* property), 134
- `types()` (*BashIntermedSymbols* property), 288
- `types()` (*IntermediateSymbolTable* property), 383
- `types()` (*ISFormatTable* property), 380
- `types()` (*LinuxKernelIntermedSymbols* property), 256
- `types()` (*MacKernelIntermedSymbols* property), 291
- `types()` (*NativeTable* property), 406
- `types()` (*NativeTableInterface* property), 135
- `types()` (*SymbolTableInterface* property), 139
- `types()` (*Version1Format* property), 385
- `types()` (*Version2Format* property), 388
- `types()` (*Version3Format* property), 391
- `types()` (*Version4Format* property), 393



types() (*Version5Format* property), 396  
 types() (*Version6Format* property), 398  
 types() (*Version7Format* property), 401  
 types() (*Version8Format* property), 404  
 types() (*WindowsKernelIntermedSymbols* property), 315

## U

UNICODE\_STRING (class in *volatility.framework.symbols.windows.extensions*), 343  
 UNICODE\_STRING.VolTemplateProxy (class in *volatility.framework.symbols.windows.extensions*), 343  
 UnionType (class in *volatility.framework.objects*), 231  
 UnionType.VolTemplateProxy (class in *volatility.framework.objects*), 231  
 unixtime\_to\_datetime() (in module *volatility.framework.renderers.conversion*), 240  
 UnparsableValue (class in *volatility.framework.renderers*), 240  
 UnreadableValue (class in *volatility.framework.renderers*), 240  
 unsatisfied() (*AutomagicInterface* class method), 100  
 unsatisfied() (*Banners* class method), 534  
 unsatisfied() (*Bash* class method), 410, 434  
 unsatisfied() (*BashIntermedSymbols* class method), 288  
 unsatisfied() (*BigPools* class method), 478  
 unsatisfied() (*BooleanRequirement* method), 71  
 unsatisfied() (*BufferDataLayer* class method), 178  
 unsatisfied() (*BytesRequirement* method), 72  
 unsatisfied() (*Check\_afinfo* class method), 411  
 unsatisfied() (*Check\_creds* class method), 413  
 unsatisfied() (*Check\_idt* class method), 414  
 unsatisfied() (*Check\_modules* class method), 416  
 unsatisfied() (*Check\_syscall* class method), 418, 436  
 unsatisfied() (*Check\_sysctl* class method), 437  
 unsatisfied() (*Check\_trap\_table* class method), 439  
 unsatisfied() (*ChoiceRequirement* method), 74  
 unsatisfied() (*ClassRequirement* method), 103  
 unsatisfied() (*CmdLine* class method), 480  
 unsatisfied() (*ComplexListRequirement* method), 75  
 unsatisfied() (*ConfigurableInterface* class method), 104  
 unsatisfied() (*ConfigurableRequirementInterface* method), 105  
 unsatisfied() (*ConfigWriter* class method), 536  
 unsatisfied() (*ConstructableRequirementInterface* method), 107  
 unsatisfied() (*ConstructionMagic* class method), 47  
 unsatisfied() (*DataLayerInterface* class method), 118  
 unsatisfied() (*DllList* class method), 482  
 unsatisfied() (*DriverIrp* class method), 484  
 unsatisfied() (*DriverScan* class method), 485  
 unsatisfied() (*DumpFiles* class method), 488  
 unsatisfied() (*Elf64Layer* class method), 149  
 unsatisfied() (*Elfs* class method), 419  
 unsatisfied() (*Envvars* class method), 489  
 unsatisfied() (*FileLayer* class method), 180  
 unsatisfied() (*FileScan* class method), 491  
 unsatisfied() (*FrameworkInfo* class method), 537  
 unsatisfied() (*GetServiceSIDs* class method), 492  
 unsatisfied() (*GetSIDs* class method), 494  
 unsatisfied() (*Handles* class method), 496  
 unsatisfied() (*HiveList* class method), 472  
 unsatisfied() (*HiveScan* class method), 473  
 unsatisfied() (*Ifconfig* class method), 440  
 unsatisfied() (*Info* class method), 498  
 unsatisfied() (*Intel* class method), 152  
 unsatisfied() (*Intel32e* class method), 154  
 unsatisfied() (*IntelPAE* class method), 156  
 unsatisfied() (*IntermediateSymbolTable* class method), 383  
 unsatisfied() (*IntRequirement* method), 77  
 unsatisfied() (*IsfInfo* class method), 539  
 unsatisfied() (*ISFormatTable* class method), 380  
 unsatisfied() (*Kauth\_listeners* class method), 442  
 unsatisfied() (*Kauth\_scopes* class method), 443  
 unsatisfied() (*KernelPDBScanner* class method), 57  
 unsatisfied() (*Kevents* class method), 445  
 unsatisfied() (*Keyboard\_notifiers* class method), 421  
 unsatisfied() (*LayerListRequirement* method), 79  
 unsatisfied() (*LayerStacker* class method), 60  
 unsatisfied() (*LayerWriter* class method), 541  
 unsatisfied() (*LimeLayer* class method), 168  
 unsatisfied() (*LinearlyMappedLayer* class method), 171  
 unsatisfied() (*LinuxBannerCache* class method), 48  
 unsatisfied() (*LinuxKernelIntermedSymbols* class method), 256  
 unsatisfied() (*LinuxSymbolFinder* class method), 50  
 unsatisfied() (*List\_Files* class method), 447  
 unsatisfied() (*ListRequirement* method), 80  
 unsatisfied() (*Lsmmod* class method), 423, 449  
 unsatisfied() (*Lsof* class method), 424, 450

`unsatisfied()` (*MacBannerCache* class method), 52  
`unsatisfied()` (*MacKernelIntermedSymbols* class method), 291  
`unsatisfied()` (*MacSymbolFinder* class method), 54  
`unsatisfied()` (*Malfind* class method), 426, 451, 500  
`unsatisfied()` (*Maps* class method), 427, 456  
`unsatisfied()` (*Memmap* class method), 502  
`unsatisfied()` (*ModScan* class method), 504  
`unsatisfied()` (*Modules* class method), 506  
`unsatisfied()` (*Mount* class method), 453  
`unsatisfied()` (*MultiRequirement* method), 82  
`unsatisfied()` (*MutantScan* class method), 508  
`unsatisfied()` (*NetScan* class method), 511  
`unsatisfied()` (*Netstat* class method), 455  
`unsatisfied()` (*NonLinearlySegmentedLayer* class method), 190  
`unsatisfied()` (*PdbMSFStream* class method), 173  
`unsatisfied()` (*PdbMultiStreamFormat* class method), 176  
`unsatisfied()` (*PluginInterface* class method), 129  
`unsatisfied()` (*PluginRequirement* method), 83  
`unsatisfied()` (*PoolScanner* class method), 514  
`unsatisfied()` (*PrintKey* class method), 475  
`unsatisfied()` (*Privs* class method), 515  
`unsatisfied()` (*Psaux* class method), 458  
`unsatisfied()` (*PsList* class method), 429, 461, 518  
`unsatisfied()` (*PsScan* class method), 520  
`unsatisfied()` (*PsTree* class method), 431, 463, 522  
`unsatisfied()` (*QemuSuspendLayer* class method), 183  
`unsatisfied()` (*RegistryHive* class method), 186  
`unsatisfied()` (*RequirementInterface* method), 110  
`unsatisfied()` (*SegmentedLayer* class method), 192  
`unsatisfied()` (*SimpleTypeRequirement* method), 111  
`unsatisfied()` (*Socket\_filters* class method), 464  
`unsatisfied()` (*SSDT* class method), 523  
`unsatisfied()` (*StringRequirement* method), 85  
`unsatisfied()` (*Strings* class method), 525  
`unsatisfied()` (*SymbolBannerCache* class method), 62  
`unsatisfied()` (*SymbolFinder* class method), 63  
`unsatisfied()` (*SymbolTableInterface* class method), 139  
`unsatisfied()` (*SymbolTableRequirement* method), 86  
`unsatisfied()` (*SymlinkScan* class method), 527  
`unsatisfied()` (*Timeliner* class method), 543  
`unsatisfied()` (*Timers* class method), 466  
`unsatisfied()` (*TranslationLayerInterface* class method), 122  
`unsatisfied()` (*TranslationLayerRequirement* method), 88  
`unsatisfied()` (*Trustedbsd* class method), 467  
`unsatisfied()` (*tty\_check* class method), 432  
`unsatisfied()` (*URIRequirement* method), 90  
`unsatisfied()` (*UserAssist* class method), 477  
`unsatisfied()` (*VadInfo* class method), 529  
`unsatisfied()` (*VerInfo* class method), 531  
`unsatisfied()` (*Version1Format* class method), 385  
`unsatisfied()` (*Version2Format* class method), 388  
`unsatisfied()` (*Version3Format* class method), 391  
`unsatisfied()` (*Version4Format* class method), 393  
`unsatisfied()` (*Version5Format* class method), 396  
`unsatisfied()` (*Version6Format* class method), 398  
`unsatisfied()` (*Version7Format* class method), 401  
`unsatisfied()` (*Version8Format* class method), 404  
`unsatisfied()` (*VersionRequirement* method), 91  
`unsatisfied()` (*VFSevents* class method), 469  
`unsatisfied()` (*VirtMap* class method), 533  
`unsatisfied()` (*VmwareLayer* class method), 195  
`unsatisfied()` (*Volshell* class method), 33, 36, 38, 40  
`unsatisfied()` (*WindowsCrashDump32Layer* class method), 143  
`unsatisfied()` (*WindowsCrashDump64Layer* class method), 146  
`unsatisfied()` (*WindowsIntel* class method), 158  
`unsatisfied()` (*WindowsIntel32e* class method), 161  
`unsatisfied()` (*WindowsIntelPAE* class method), 163  
`unsatisfied()` (*WindowsKernelIntermedSymbols* class method), 315  
`unsatisfied()` (*WindowsMixin* class method), 165  
`unsatisfied()` (*WinSwapLayers* class method), 67  
`unsatisfied()` (*WintelHelper* class method), 69  
`unsatisfied_children()` (*BooleanRequirement* method), 71  
`unsatisfied_children()` (*BytesRequirement* method), 72  
`unsatisfied_children()` (*ChoiceRequirement* method), 74  
`unsatisfied_children()` (*ClassRequirement* method), 103  
`unsatisfied_children()` (*ComplexListRequirement* method), 75  
`unsatisfied_children()` (*ConfigurableRequirementInterface* method), 105  
`unsatisfied_children()` (*ConstructableRequirementInterface* method), 107  
`unsatisfied_children()` (*IntRequirement* method), 77  
`unsatisfied_children()` (*LayerListRequirement* method), 79  
`unsatisfied_children()` (*ListRequirement* method), 80  
`unsatisfied_children()` (*MultiRequirement* method), 82

- `unsatisfied_children()` (*PluginRequirement method*), 83
  - `unsatisfied_children()` (*RequirementInterface method*), 110
  - `unsatisfied_children()` (*SimpleTypeRequirement method*), 111
  - `unsatisfied_children()` (*StringRequirement method*), 85
  - `unsatisfied_children()` (*SymbolTableRequirement method*), 86
  - `unsatisfied_children()` (*TranslationLayerRequirement method*), 88
  - `unsatisfied_children()` (*URIRequirement method*), 90
  - `unsatisfied_children()` (*VersionRequirement method*), 91
  - `UnsatisfiedException`, 408
  - `update_vol()` (*ObjectTemplate method*), 235
  - `update_vol()` (*ReferenceTemplate method*), 236
  - `update_vol()` (*SymbolSpace.UnresolvedTemplate method*), 250
  - `update_vol()` (*Template method*), 126
  - `upper()` (*Bytes method*), 207
  - `upper()` (*HexBytes method*), 246
  - `upper()` (*MultiTypeData method*), 249
  - `upper()` (*String method*), 229
  - `URIRequirement` (class in *volatility.framework.configuration.requirements*), 88
  - `UserAssist` (class in *volatility.plugins.windows.registry.userassist*), 476
  - `uses_cache()` (*ResourceAccessor method*), 188
- ## V
- `VACB` (class in *volatility.framework.symbols.windows.extensions*), 344
  - `VACB.VolTemplateProxy` (class in *volatility.framework.symbols.windows.extensions*), 345
  - `VACB_ARRAY` (*SHARED\_CACHE\_MAP attribute*), 339
  - `VACB_BLOCK` (*SHARED\_CACHE\_MAP attribute*), 339
  - `VACB_LEVEL_SHIFT` (*SHARED\_CACHE\_MAP attribute*), 339
  - `VACB_OFFSET_SHIFT` (*SHARED\_CACHE\_MAP attribute*), 339
  - `VACB_SIZE_OF_FIRST_LEVEL` (*SHARED\_CACHE\_MAP attribute*), 339
  - `vad_dump()` (*VadInfo class method*), 529
  - `VadInfo` (class in *volatility.plugins.windows.vadinfo*), 528
  - `valid()` (in module *volatility.schemas*), 543
  - `validate()` (in module *volatility.schemas*), 543
  - `VALIDDUMP` (*WindowsCrashDump32Layer attribute*), 141
  - `VALIDDUMP` (*WindowsCrashDump64Layer attribute*), 144
  - `values()` (*HierarchicalDict method*), 109
  - `values()` (*LayerContainer method*), 119
  - `values()` (*ObjectInformation method*), 123
  - `values()` (*ReadOnlyMapping method*), 125
  - `values()` (*SymbolSpace method*), 251
  - `values()` (*SymbolSpaceInterface method*), 137
  - `values()` (*TreeGrid method*), 131, 239
  - `values()` (*TreeNode property*), 132, 239
  - `VerInfo` (class in *volatility.plugins.windows.verinfo*), 530
  - `version` (*Banners attribute*), 535
  - `version` (*Bash attribute*), 410, 434
  - `version` (*BigPools attribute*), 479
  - `version` (*BytesScanner attribute*), 140
  - `version` (*Check\_afinfo attribute*), 412
  - `version` (*Check\_creds attribute*), 413
  - `version` (*Check\_idt attribute*), 415
  - `version` (*Check\_modules attribute*), 416
  - `version` (*Check\_syscall attribute*), 418, 436
  - `version` (*Check\_sysctl attribute*), 437
  - `version` (*Check\_trap\_table attribute*), 439
  - `version` (*CmdLine attribute*), 480
  - `version` (*ConfigWriter attribute*), 536
  - `version` (*DllList attribute*), 482
  - `version` (*DriverIrp attribute*), 484
  - `version` (*DriverScan attribute*), 486
  - `version` (*DumpFiles attribute*), 488
  - `version` (*Elfs attribute*), 419
  - `version` (*Envars attribute*), 489
  - `version` (*FileScan attribute*), 491
  - `version` (*FrameworkInfo attribute*), 538
  - `version` (*GetServiceSIDs attribute*), 493
  - `version` (*GetSIDs attribute*), 494
  - `version` (*Handles attribute*), 496
  - `version` (*HiveList attribute*), 472
  - `version` (*HiveScan attribute*), 474
  - `version` (*Ifconfig attribute*), 440
  - `version` (*Info attribute*), 498
  - `version` (*IsfInfo attribute*), 539
  - `version` (*ISFormatTable attribute*), 380
  - `version` (*Kauth\_listeners attribute*), 442
  - `version` (*Kauth\_scopes attribute*), 444
  - `version` (*Kevents attribute*), 445
  - `version` (*Keyboard\_notifiers attribute*), 421
  - `version` (*LayerWriter attribute*), 541
  - `VERSION` (*LimeLayer attribute*), 166
  - `version` (*LinuxUtilities attribute*), 256
  - `version` (*List\_Files attribute*), 447
  - `version` (*Lsmmod attribute*), 423, 449
  - `version` (*Lsof attribute*), 424, 450

`version` (*MacUtilities* attribute), 291  
`version` (*Malfind* attribute), 426, 452, 500  
`version` (*Maps* attribute), 427, 457  
`version` (*Memmap* attribute), 502  
`version` (*ModScan* attribute), 504  
`version` (*Modules* attribute), 506  
`version` (*Mount* attribute), 453  
`version` (*MultiStringScanner* attribute), 140  
`version` (*MutantScan* attribute), 508  
`version` (*NetScan* attribute), 511  
`version` (*Netstat* attribute), 455  
`version` (*PageMapScanner* attribute), 66  
`version` (*PdbSignatureScanner* attribute), 377  
`version` (*PluginInterface* attribute), 129  
`version` (*PoolHeaderScanner* attribute), 511  
`version` (*PoolScanner* attribute), 514  
`version` (*PrintKey* attribute), 475  
`version` (*Privs* attribute), 515  
`version` (*Psaux* attribute), 458  
`version` (*PsList* attribute), 429, 462, 518  
`version` (*PsScan* attribute), 520  
`version` (*PsTree* attribute), 431, 463, 522  
`version` (*RegexScanner* attribute), 140  
`version` (*ScannerInterface* attribute), 120  
`version` (*Socket\_filters* attribute), 464  
`version` (*SSDT* attribute), 524  
`version` (*Strings* attribute), 525  
`version` (*SymlinkScan* attribute), 527  
`version` (*Timeliner* attribute), 543  
`version` (*Timers* attribute), 466  
`version` (*Trustedbsd* attribute), 467  
`version` (*tty\_check* attribute), 432  
`version` (*UserAssist* attribute), 477  
`version` (*VadInfo* attribute), 530  
`version` (*VerInfo* attribute), 531  
`version` (*Version1Format* attribute), 386  
`version` (*Version2Format* attribute), 388  
`version` (*Version3Format* attribute), 391  
`version` (*Version4Format* attribute), 393  
`version` (*Version5Format* attribute), 396  
`version` (*Version6Format* attribute), 399  
`version` (*Version7Format* attribute), 401  
`version` (*Version8Format* attribute), 404  
`version` (*VersionableInterface* attribute), 112  
`version` (*VFSevents* attribute), 469  
`version` (*VirtMap* attribute), 533  
`version` (*Volshell* attribute), 34, 36, 38, 40  
`Version1Format` (class in *volatility.framework.symbols.intermed*), 383  
`Version2Format` (class in *volatility.framework.symbols.intermed*), 386  
`Version3Format` (class in *volatility.framework.symbols.intermed*), 388  
`Version4Format` (class in *volatility.framework.symbols.intermed*), 391  
`Version5Format` (class in *volatility.framework.symbols.intermed*), 393  
`Version6Format` (class in *volatility.framework.symbols.intermed*), 396  
`Version7Format` (class in *volatility.framework.symbols.intermed*), 399  
`Version8Format` (class in *volatility.framework.symbols.intermed*), 401  
`VersionableInterface` (class in *volatility.framework.interfaces.configuration*), 112  
`VersionRequirement` (class in *volatility.framework.configuration.requirements*), 90  
`VFSevents` (class in *volatility.plugins.mac.vfsevents*), 468  
`vfsmount` (class in *volatility.framework.symbols.linux.extensions*), 275  
`vfsmount.VolTemplateProxy` (class in *volatility.framework.symbols.linux.extensions*), 275  
`VirtMap` (class in *volatility.plugins.windows.virtmap*), 532  
`virtual_process_from_physical()` (*PsScan* class method), 520  
`virtual_to_physical_address()` (*LinuxIntelStacker* class method), 49  
`virtual_to_physical_address()` (*MacIntelStacker* class method), 53  
`visit()` (*TreeGrid* method), 131, 239  
`visit_nodes()` (*RegistryHive* method), 187  
`vm_area_struct` (class in *volatility.framework.symbols.linux.extensions*), 277  
`vm_area_struct.VolTemplateProxy` (class in *volatility.framework.symbols.linux.extensions*), 277  
`vm_map_entry` (class in *volatility.framework.symbols.mac.extensions*), 307  
`vm_map_entry.VolTemplateProxy` (class in *volatility.framework.symbols.mac.extensions*), 308  
`vm_map_object` (class in *volatility.framework.symbols.mac.extensions*), 309  
`vm_map_object.VolTemplateProxy` (class in *volatility.framework.symbols.mac.extensions*), 309  
`VmwareFormatException`, 193  
`VmwareLayer` (class in *volatility.framework.layers.vmware*), 193  
`VmwareStacker` (class in *volatility.framework.layers.vmware*), 195  
`vnode` (class in *volatility*



`ity.framework.symbols.mac.extensions`), 310  
`vnode.VolTemplateProxy` (class in `volatility.framework.symbols.mac.extensions`), 311  
`vnode_filters` (Kevents attribute), 445  
`Void` (class in `volatility.framework.objects`), 232  
`Void.VolTemplateProxy` (class in `volatility.framework.objects`), 233  
`vol()` (AggregateType property), 197  
`vol()` (Array property), 198  
`vol()` (BitField property), 200  
`vol()` (Boolean property), 203  
`vol()` (Bytes property), 207  
`vol()` (Char property), 210  
`vol()` (ClassType property), 211  
`vol()` (CM\_KEY\_BODY property), 364  
`vol()` (CM\_KEY\_NODE property), 365  
`vol()` (CM\_KEY\_VALUE property), 367  
`vol()` (CMHIVE property), 362  
`vol()` (CONTROL\_AREA property), 317  
`vol()` (dentry property), 258  
`vol()` (DEVICE\_OBJECT property), 318  
`vol()` (DRIVER\_OBJECT property), 320  
`vol()` (elf property), 282  
`vol()` (elf\_phdr property), 283  
`vol()` (elf\_sym property), 285  
`vol()` (Enumeration property), 214  
`vol()` (EPROCESS property), 322  
`vol()` (ETHREAD property), 324  
`vol()` (EX\_FAST\_REF property), 325  
`vol()` (ExecutiveObject property), 353  
`vol()` (FILE\_OBJECT property), 327  
`vol()` (fileglob property), 293  
`vol()` (files\_struct property), 259  
`vol()` (Float property), 216  
`vol()` (fs\_struct property), 261  
`vol()` (Function property), 218  
`vol()` (GenericIntelProcess property), 253  
`vol()` (hist\_entry property), 280  
`vol()` (HMAP\_ENTRY property), 369  
`vol()` (ifnet property), 295  
`vol()` (IMAGE\_DOS\_HEADER property), 350  
`vol()` (IMAGE\_NT\_HEADERS property), 351  
`vol()` (inpcb property), 296  
`vol()` (Integer property), 220  
`vol()` (kauth\_scope property), 298  
`vol()` (KDDEBUGGER\_DATA64 property), 347  
`vol()` (KMUTANT property), 329  
`vol()` (kobject property), 262  
`vol()` (KSYSTEM\_TIME property), 330  
`vol()` (KTHREAD property), 332  
`vol()` (LIST\_ENTRY property), 333  
`vol()` (list\_head property), 264  
`vol()` (mm\_struct property), 265  
`vol()` (MMVAD property), 335  
`vol()` (MMVAD\_SHORT property), 337  
`vol()` (module property), 267  
`vol()` (mount property), 269  
`vol()` (OBJECT\_HEADER property), 355  
`vol()` (OBJECT\_SYMBOLIC\_LINK property), 339  
`vol()` (ObjectInterface property), 124  
`vol()` (ObjectTemplate property), 235  
`vol()` (Pointer property), 222  
`vol()` (POOL\_HEADER property), 357  
`vol()` (POOL\_HEADER\_VISTA property), 358  
`vol()` (POOL\_TRACKER\_BIG\_PAGES property), 360  
`vol()` (PrimitiveObject property), 224  
`vol()` (proc property), 299  
`vol()` (qstr property), 270  
`vol()` (queue\_entry property), 301  
`vol()` (ReferenceTemplate property), 236  
`vol()` (SERVICE\_HEADER property), 371  
`vol()` (SERVICE\_RECORD property), 373  
`vol()` (SHARED\_CACHE\_MAP property), 341  
`vol()` (sockaddr property), 303  
`vol()` (sockaddr\_dl property), 304  
`vol()` (socket property), 306  
`vol()` (String property), 229  
`vol()` (struct\_file property), 272  
`vol()` (StructType property), 231  
`vol()` (super\_block property), 273  
`vol()` (SymbolSpace.UnresolvedTemplate property), 250  
`vol()` (sysctl\_oid property), 307  
`vol()` (task\_struct property), 275  
`vol()` (Template property), 126  
`vol()` (TOKEN property), 343  
`vol()` (UNICODE\_STRING property), 344  
`vol()` (UnionType property), 232  
`vol()` (VACB property), 346  
`vol()` (vfsmount property), 276  
`vol()` (vm\_area\_struct property), 278  
`vol()` (vm\_map\_entry property), 309  
`vol()` (vm\_map\_object property), 310  
`vol()` (vnode property), 312  
`vol()` (Void property), 234  
`volatility`  
    module, 27  
`volatility.cli`  
    module, 28  
`volatility.cli.text_renderer`  
    module, 41  
`volatility.cli.volargparse`  
    module, 43  
`volatility.cli.volshell`  
    module, 29  
`volatility.cli.volshell.generic`  
    module, 30  
`volatility.cli.volshell.linux`

- module, 34
- volatility.cli.volshell.mac
  - module, 36
- volatility.cli.volshell.windows
  - module, 38
- volatility.framework
  - module, 44
- volatility.framework.automagic
  - module, 45
- volatility.framework.automagic.constructors
  - module, 46
- volatility.framework.automagic.linux
  - module, 47
- volatility.framework.automagic.mac
  - module, 51
- volatility.framework.automagic.pdbscan
  - module, 54
- volatility.framework.automagic.stacker
  - module, 58
- volatility.framework.automagic.symbol\_cache
  - module, 60
- volatility.framework.automagic.symbol\_finder
  - module, 62
- volatility.framework.automagic.windows
  - module, 64
- volatility.framework.configuration
  - module, 70
- volatility.framework.configuration.requirements
  - module, 70
- volatility.framework.constants
  - module, 92
- volatility.framework.constants.linux
  - module, 93
- volatility.framework.constants.windows
  - module, 93
- volatility.framework.contexts
  - module, 93
- volatility.framework.exceptions
  - module, 406
- volatility.framework.interfaces
  - module, 99
- volatility.framework.interfaces.automagic
  - module, 99
- volatility.framework.interfaces.configuration
  - module, 101
- volatility.framework.interfaces.context
  - module, 112
- volatility.framework.interfaces.layers
  - module, 116
- volatility.framework.interfaces.objects
  - module, 122
- volatility.framework.interfaces.plugins
  - module, 126
- volatility.framework.interfaces.renderers
  - module, 129
- volatility.framework.interfaces.symbols
  - module, 132
- volatility.framework.layers
  - module, 140
- volatility.framework.layers.codecs
  - module, 140
- volatility.framework.layers.crash
  - module, 141
- volatility.framework.layers.elf
  - module, 147
- volatility.framework.layers.intel
  - module, 150
- volatility.framework.layers.lime
  - module, 166
- volatility.framework.layers.linear
  - module, 169
- volatility.framework.layers.msfs
  - module, 171
- volatility.framework.layers.physical
  - module, 176
- volatility.framework.layers.qemu
  - module, 181
- volatility.framework.layers.registry
  - module, 184
- volatility.framework.layers.resources
  - module, 187
- volatility.framework.layers.scanners
  - module, 140
- volatility.framework.layers.scanners.multiregexp
  - module, 141
- volatility.framework.layers.segmented
  - module, 188
- volatility.framework.layers.vmware
  - module, 193
- volatility.framework.objects
  - module, 195
- volatility.framework.objects.templates
  - module, 234
- volatility.framework.objects.utility
  - module, 236
- volatility.framework.plugins
  - module, 236
- volatility.framework.renderers
  - module, 237
- volatility.framework.renderers.conversion
  - module, 240
- volatility.framework.renderers.format\_hints
  - module, 241
- volatility.framework.symbols
  - module, 249
- volatility.framework.symbols.generic
  - module, 252
- volatility.framework.symbols.intermediary
  - module, 252

module, 377	module, 540
volatility.framework.symbols.linux	volatility.plugins.linux
module, 253	module, 408
volatility.framework.symbols.linux.bash	volatility.plugins.linux.bash
module, 285	module, 409
volatility.framework.symbols.linux.extensions	volatility.plugins.linux.check_afinfo
module, 257	module, 410
volatility.framework.symbols.linux.extensions.bash	volatility.plugins.linux.check_creds
module, 279	module, 412
volatility.framework.symbols.linux.extensions.bash.elf	volatility.plugins.linux.check_idt
module, 280	module, 413
volatility.framework.symbols.mac	volatility.plugins.linux.check_modules
module, 288	module, 415
volatility.framework.symbols.mac.extensions	volatility.plugins.linux.check_syscall
module, 292	module, 416
volatility.framework.symbols.metadata	volatility.plugins.linux.elfs
module, 404	module, 418
volatility.framework.symbols.native	volatility.plugins.linux.keyboard_notifiers
module, 405	module, 420
volatility.framework.symbols.windows	volatility.plugins.linux.lsmmod
module, 312	module, 421
volatility.framework.symbols.windows.extensions	volatility.plugins.linux.lsof
module, 315	module, 423
volatility.framework.symbols.windows.extensions.kdbg	volatility.plugins.linux.malfind
module, 346	module, 424
volatility.framework.symbols.windows.extensions.network	volatility.plugins.linux.proc
module, 348	module, 426
volatility.framework.symbols.windows.extensions.ps	volatility.plugins.linux.pslist
module, 348	module, 427
volatility.framework.symbols.windows.extensions.pspop	volatility.plugins.linux.pstree
module, 352	module, 429
volatility.framework.symbols.windows.extensions.registry	volatility.plugins.linux.tty_check
module, 361	module, 431
volatility.framework.symbols.windows.extensions.sysplugins	volatility.plugins.mac
module, 370	module, 433
volatility.framework.symbols.windows.pdbhelper	volatility.plugins.mac.bash
module, 373	module, 433
volatility.framework.symbols.windows.pdbutils	volatility.plugins.mac.check_syscall
module, 375	module, 435
volatility.framework.symbols.windows.verdict	volatility.plugins.mac.check_sysctl
module, 377	module, 436
volatility.framework.symbols.wrappers	volatility.plugins.mac.check_trap_table
module, 406	module, 438
volatility.plugins	volatility.plugins.mac.ifconfig
module, 408	module, 439
volatility.plugins.banners	volatility.plugins.mac.kauth_listeners
module, 533	module, 441
volatility.plugins.configwriter	volatility.plugins.mac.kauth_scopes
module, 535	module, 442
volatility.plugins.frameworkinfo	volatility.plugins.mac.kevents
module, 536	module, 444
volatility.plugins.isfinfo	volatility.plugins.mac.list_files
module, 538	module, 446
volatility.plugins.layerwriter	volatility.plugins.mac.lsmmod

- module, 447
- volatility.plugins.mac.lsof
  - module, 449
- volatility.plugins.mac.malfind
  - module, 450
- volatility.plugins.mac.mount
  - module, 452
- volatility.plugins.mac.netstat
  - module, 454
- volatility.plugins.mac.proc\_maps
  - module, 455
- volatility.plugins.mac.psaux
  - module, 457
- volatility.plugins.mac.pslist
  - module, 459
- volatility.plugins.mac.pstree
  - module, 462
- volatility.plugins.mac.socket\_filters
  - module, 463
- volatility.plugins.mac.timers
  - module, 465
- volatility.plugins.mac.trustedbsd
  - module, 466
- volatility.plugins.mac.vfsevents
  - module, 468
- volatility.plugins.timeliner
  - module, 542
- volatility.plugins.windows
  - module, 469
- volatility.plugins.windows.bigpools
  - module, 477
- volatility.plugins.windows.cmdline
  - module, 479
- volatility.plugins.windows.dlllist
  - module, 481
- volatility.plugins.windows.driverirp
  - module, 483
- volatility.plugins.windows.driverscan
  - module, 484
- volatility.plugins.windows.dumpfiles
  - module, 486
- volatility.plugins.windows.envvars
  - module, 488
- volatility.plugins.windows.filescan
  - module, 490
- volatility.plugins.windows.getservicesids
  - module, 491
- volatility.plugins.windows.getsids
  - module, 493
- volatility.plugins.windows.handles
  - module, 494
- volatility.plugins.windows.info
  - module, 496
- volatility.plugins.windows.malfind
  - module, 499
- volatility.plugins.windows.memmap
  - module, 501
- volatility.plugins.windows.modscan
  - module, 502
- volatility.plugins.windows.modules
  - module, 504
- volatility.plugins.windows.mutantscan
  - module, 507
- volatility.plugins.windows.netscan
  - module, 508
- volatility.plugins.windows.poolscanner
  - module, 511
- volatility.plugins.windows.privileges
  - module, 514
- volatility.plugins.windows.pslist
  - module, 516
- volatility.plugins.windows.psscan
  - module, 518
- volatility.plugins.windows.pstree
  - module, 521
- volatility.plugins.windows.registry
  - module, 469
- volatility.plugins.windows.registry.hivelist
  - module, 470
- volatility.plugins.windows.registry.hivescan
  - module, 472
- volatility.plugins.windows.registry.printkey
  - module, 474
- volatility.plugins.windows.registry.userassist
  - module, 476
- volatility.plugins.windows.ssdt
  - module, 522
- volatility.plugins.windows.strings
  - module, 524
- volatility.plugins.windows.symlinkscan
  - module, 526
- volatility.plugins.windows.vadinfo
  - module, 528
- volatility.plugins.windows.verinfo
  - module, 530
- volatility.plugins.windows.virtmap
  - module, 532
- volatility.schemas
  - module, 543
- volatility.symbols
  - module, 544
- VolatilityException, 408
- VolShell (*class in volatility.cli.volshell*), 29
- Volshell (*class in volatility.cli.volshell.generic*), 31
- Volshell (*class in volatility.cli.volshell.linux*), 34
- Volshell (*class in volatility.cli.volshell.mac*), 36
- Volshell (*class in volatility.cli.volshell.windows*), 38

## W

- walk\_internal\_list() (*LinuxUtilities* class method), 256
- walk\_list() (*queue\_entry* method), 301
- walk\_list\_head() (*MacUtilities* class method), 292
- walk\_slist() (*MacUtilities* class method), 292
- walk\_tailq() (*MacUtilities* class method), 292
- WarningFindSpec (class in *volatility*), 27
- WindowsCrashDump32Layer (class in *volatility.framework.layers.crash*), 141
- WindowsCrashDump64Layer (class in *volatility.framework.layers.crash*), 143
- WindowsCrashDumpFormatException, 146
- WindowsCrashDumpStacker (class in *volatility.framework.layers.crash*), 146
- WindowsIntel (class in *volatility.framework.layers.intel*), 156
- WindowsIntel32e (class in *volatility.framework.layers.intel*), 159
- WindowsIntelPAE (class in *volatility.framework.layers.intel*), 161
- WindowsIntelStacker (class in *volatility.framework.automagic.windows*), 68
- WindowsKernelIntermedSymbols (class in *volatility.framework.symbols.windows*), 312
- WindowsMetadata (class in *volatility.framework.symbols.metadata*), 404
- WindowsMixin (class in *volatility.framework.layers.intel*), 163
- WinSwapLayers (class in *volatility.framework.automagic.windows*), 66
- WintelHelper (class in *volatility.framework.automagic.windows*), 68
- wintime\_to\_datetime() (in module *volatility.framework.renderers.conversion*), 240
- with\_traceback() (*ElfFormatException* method), 149
- with\_traceback() (*InvalidAddressException* method), 406
- with\_traceback() (*LayerException* method), 407
- with\_traceback() (*LimeFormatException* method), 166
- with\_traceback() (*MissingModuleException* method), 407
- with\_traceback() (*PagedInvalidAddressException* method), 407
- with\_traceback() (*PDBFormatException* method), 171
- with\_traceback() (*PluginRequirementException* method), 407
- with\_traceback() (*PluginVersionException* method), 407
- with\_traceback() (*RegistryFormatException* method), 184
- with\_traceback() (*RegistryInvalidIndex* method), 187
- with\_traceback() (*SwappedInvalidAddressException* method), 407
- with\_traceback() (*SymbolError* method), 408
- with\_traceback() (*SymbolSpaceError* method), 408
- with\_traceback() (*UnsatisfiedException* method), 408
- with\_traceback() (*VmwareFormatException* method), 193
- with\_traceback() (*VolatilityException* method), 408
- with\_traceback() (*WindowsCrashDumpFormatException* method), 146
- writable() (*FileHandlerInterface* method), 127
- writable() (*NullFileHandler* method), 31
- write() (*AggregateType* method), 197
- write() (*Array* method), 198
- write() (*BitField* method), 201
- write() (*Boolean* method), 203
- write() (*BufferDataLayer* method), 178
- write() (*Bytes* method), 207
- write() (*Char* method), 210
- write() (*ClassType* method), 211
- write() (*CM\_KEY\_BODY* method), 364
- write() (*CM\_KEY\_NODE* method), 365
- write() (*CM\_KEY\_VALUE* method), 367
- write() (*CMHIVE* method), 362
- write() (*CONTROL\_AREA* method), 317
- write() (*DataLayerInterface* method), 118
- write() (*dentry* method), 258
- write() (*DEVICE\_OBJECT* method), 318
- write() (*DRIVER\_OBJECT* method), 320
- write() (*elf* method), 282
- write() (*Elf64Layer* method), 149
- write() (*elf\_phdr* method), 283
- write() (*elf\_sym* method), 285
- write() (*Enumeration* method), 214
- write() (*EPROCESS* method), 322
- write() (*ETHREAD* method), 324
- write() (*EX\_FAST\_REF* method), 325
- write() (*ExecutiveObject* method), 353
- write() (*FILE\_OBJECT* method), 327
- write() (*fileglob* method), 293
- write() (*FileHandlerInterface* method), 127
- write() (*FileLayer* method), 180
- write() (*files\_struct* method), 260
- write() (*Float* method), 216
- write() (*fs\_struct* method), 261
- write() (*Function* method), 218
- write() (*GenericIntelProcess* method), 253
- write() (*hist\_entry* method), 280
- write() (*HMAP\_ENTRY* method), 369



`write()` (*ifnet method*), 295  
`write()` (*IMAGE\_DOS\_HEADER method*), 350  
`write()` (*IMAGE\_NT\_HEADERS method*), 351  
`write()` (*inpcb method*), 296  
`write()` (*Integer method*), 220  
`write()` (*Intel method*), 152  
`write()` (*Intel32e method*), 154  
`write()` (*IntelPAE method*), 156  
`write()` (*kauth\_scope method*), 298  
`write()` (*KDDEBUGGER\_DATA64 method*), 347  
`write()` (*KMUTANT method*), 329  
`write()` (*kobject method*), 262  
`write()` (*KSYSTEM\_TIME method*), 330  
`write()` (*KTHREAD method*), 332  
`write()` (*LayerContainer method*), 119  
`write()` (*LimeLayer method*), 168  
`write()` (*LinearlyMappedLayer method*), 171  
`write()` (*LIST\_ENTRY method*), 333  
`write()` (*list\_head method*), 264  
`write()` (*mm\_struct method*), 265  
`write()` (*MMVAD method*), 335  
`write()` (*MMVAD\_SHORT method*), 337  
`write()` (*module method*), 267  
`write()` (*mount method*), 269  
`write()` (*NonLinearlySegmentedLayer method*), 190  
`write()` (*NullFileHandler method*), 31  
`write()` (*OBJECT\_HEADER method*), 355  
`write()` (*OBJECT\_SYMBOLIC\_LINK method*), 339  
`write()` (*ObjectInterface method*), 124  
`write()` (*PdbMSFStream method*), 173  
`write()` (*PdbMultiStreamFormat method*), 176  
`write()` (*Pointer method*), 222  
`write()` (*POOL\_HEADER method*), 357  
`write()` (*POOL\_HEADER\_VISTA method*), 359  
`write()` (*POOL\_TRACKER\_BIG\_PAGES method*), 360  
`write()` (*PrimitiveObject method*), 224  
`write()` (*proc method*), 299  
`write()` (*QemuSuspendLayer method*), 184  
`write()` (*qstr method*), 270  
`write()` (*queue\_entry method*), 301  
`write()` (*RegistryHive method*), 187  
`write()` (*SegmentedLayer method*), 192  
`write()` (*SERVICE\_HEADER method*), 371  
`write()` (*SERVICE\_RECORD method*), 373  
`write()` (*SHARED\_CACHE\_MAP method*), 341  
`write()` (*sockaddr method*), 303  
`write()` (*sockaddr\_dl method*), 304  
`write()` (*socket method*), 306  
`write()` (*String method*), 229  
`write()` (*struct\_file method*), 272  
`write()` (*StructType method*), 231  
`write()` (*super\_block method*), 273  
`write()` (*sysctl\_oid method*), 307

`write()` (*task\_struct method*), 275  
`write()` (*TOKEN method*), 343  
`write()` (*TranslationLayerInterface method*), 122  
`write()` (*UNICODE\_STRING method*), 344  
`write()` (*UnionType method*), 232  
`write()` (*VACB method*), 346  
`write()` (*vfsmount method*), 277  
`write()` (*vm\_area\_struct method*), 278  
`write()` (*vm\_map\_entry method*), 309  
`write()` (*vm\_map\_object method*), 310  
`write()` (*VmwareLayer method*), 195  
`write()` (*vnode method*), 312  
`write()` (*Void method*), 234  
`write()` (*WindowsCrashDump32Layer method*), 143  
`write()` (*WindowsCrashDump64Layer method*), 146  
`write()` (*WindowsIntel method*), 158  
`write()` (*WindowsIntel32e method*), 161  
`write()` (*WindowsIntelPAE method*), 163  
`write()` (*WindowsMixin method*), 165  
`write_layer()` (*LayerWriter class method*), 541  
`writelines()` (*FileHandlerInterface method*), 127  
`writelines()` (*NullFileHandler method*), 31

## Z

`zfill()` (*Bytes method*), 207  
`zfill()` (*HexBytes method*), 246  
`zfill()` (*MultiTypeData method*), 249  
`zfill()` (*String method*), 229