

---

# Volatility Documentation

*Release 3.0.0<sub>alpha</sub>1*

**Volatility Foundation**

**Oct 10, 2019**



# 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 . . . . .	9
<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 Scanners . . . . .	16
4.4	Writing/Using Intermediate Symbol Format Files . . . . .	16
4.5	Writing new translation layers . . . . .	17
4.6	Writing new Templates and Objects . . . . .	17
<b>5</b>	<b>Creating New Symbol Tables</b>	<b>19</b>
5.1	How Volatility finds symbol tables . . . . .	19
5.2	Windows symbol tables . . . . .	19
5.3	Mac/Linux symbol tables . . . . .	19
<b>6</b>	<b>Python Packages</b>	<b>21</b>
6.1	volatility package . . . . .	21
<b>7</b>	<b>Indices and tables</b>	<b>427</b>

<b>Python Module Index</b>	<b>429</b>
<b>Index</b>	<b>433</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.

---

Volatility currently supports the following physical formats:

- Buffered data
- Flat/Raw files
- LiME files
- Crashdump files
- VMware snapshots (vmem and vmss)

It also supports the mappings based on the following architectures:

- Intel (x86)
- Intel PAE
- Intel 32e (x86\_64)

## 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.IntRequirement(name = 'pid',
                                                    description = "Process ID 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`

---

**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.

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.

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.

```
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.IntRequirement(name = 'pid',
                           description = "Process ID to include (all other processes_
→are excluded)",
                           optional = True)]
```

The final requirement is a Simple Requirement, populated by an integer. 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:
```

(continues on next page)

(continued from previous page)

```

    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:

- **Unreadble:** 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._file_consumer)
```

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
- any file consumers for files created during running of the plugin

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 Scanners

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

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

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

## 4.4 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]
```

## 4.5 Writing new translation layers

## 4.6 Writing new Templates and Objects



## 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.

### 5.1 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.

### 5.2 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 using a couple of different tools. The first 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.

### 5.3 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

### 6.1 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*)

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

**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: `object`

Class property decorator.

Note this will change the return type

#### 6.1.1 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: `volatility.framework.interfaces.plugins.FileConsumerInterface`

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

**consume\_file** (*filedata*)

Consumes a file as produced by a plugin.

**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, 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.

**run** ()

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

**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*)**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.

#### **consume\_file** (*filedata*)

Consumes a file as produced by a plugin.

#### **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, 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.

#### **run** ()

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

#### **main** ()

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

## Submodules

### volatility.cli.volshell.generic module

#### class NullFileConsumer

Bases: *volatility.framework.interfaces.plugins.FileConsumerInterface*

Null FileConsumer that swallows files whole

**consume\_file** (*file*)

Dummy file consumer to satisfy the FileConsumerInterface

**Return type** None

**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]]*

**consume\_file** (*file*)

Dummy file consumer to satisfy the

**Return type** None

**property context**

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

**Return type** *ContextInterface*

**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** *TreeGrid*

**classmethod get\_requirements** ()

Returns a list of Requirement objects for this plugin.

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

**help** (*\*args*)

Describes the available commands

**classmethod make\_subconfig** (*context, base\_config\_path, \*\*kwargs*)

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

**Parameters**

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

**Returns** The newly generated full configuration path

**Return type** *str*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** None

**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\_file\_consumer** (*consumer*)

Sets the file consumer 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]]`

**consume\_file(file)**

Dummy file consumer to satisfy the

**Return type** `None`

**property context**

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

**Return type** `ContextInterface`

**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** `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

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

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

**Parameters**

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

**Returns** The newly generated full configuration path

**Return type** `str`

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** None

**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\_file\_consumer** (*consumer*)

Sets the file consumer 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]]`

**consume\_file** (*file*)

Dummy file consumer to satisfy the

**Return type** `None`

**property context**

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

**Return type** `ContextInterface`

**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** `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

**classmethod make\_subconfig** (*context*, *base\_config\_path*, *\*\*kwargs*)

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

**Parameters**

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

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

**Returns** The newly generated full configuration path

**Return type** *str*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**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\_file\_consumer** (*consumer*)

Sets the file consumer 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]]*

**consume\_file** (*file*)

Dummy file consumer to satisfy the

**Return type** *None*

**property context**

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

**Return type** *ContextInterface*

**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** *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

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

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

**Parameters**

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

**Returns** The newly generated full configuration path

**Return type** *str*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**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\_file\_consumer** (*consumer*)

Sets the file consumer 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`

**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`

**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`

**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`

```
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

```
optional(func)
```

```
quoted_optional(func)
```

## 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]`

```
hide_from_subclasses(cls)
```

**Return type** `Type[+CT_co]`

```
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.

```
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.

**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** `LintelStacker`

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

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

Attempts to identify linux within this layer.

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

**stack\_order** = 12

**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** **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 `LinuxBannerCache`

**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`

**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]*

**class** **LinuxUtilities**

Bases: *object*

Class with multiple useful linux functions.

**classmethod** **aslr\_mask\_symbol\_table** (*context, symbol\_table, layer\_name, aslr\_shift=0*)

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

**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** `path_for_file` (*context, task, filp*)

Return type `str`

**classmethod** `virtual_to_physical_address` (*addr*)

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

Return type `int`

## 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** `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 `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*

**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]`

**class MacUtilities**

Bases: `object`

Class with multiple useful mac functions.

`classmethod aslr_mask_symbol_table(context, symbol_table, layer_name, aslr_shift=0)`

`classmethod files_descriptors_for_process(config, context, task)`

`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 virtual_to_physical_address(addr)`

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

Return type `int`

`walk_tailq(next_member, max_elements=4096)`

Return type `Iterable[ObjectInterface]`

**class MacintelStacker**

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

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

Attempts to identify mac within this layer.

Return type `Optional[DataLayerInterface]`

`stack_order = 12`

## 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** *Dict[str, Tuple[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\_kernels** (*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** *Dict[str, Tuple[int, Dict[str, Union[bytes, str, int, None]]]]*

**Returns** A dictionary of valid kernels

**download\_pdb\_isf** (*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`

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

**method\_kdbg\_offset** (*context, vlayer, progress\_callback=None*)

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

**method\_module\_offset** (*context, vlayer, progress\_callback=None*)

Method for finding a suitable kernel offset based on a module table.

**Return type** *Dict[str, Tuple[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\_kernels, 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\_kernels** (*Dict[str, Tuple[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\_kernels*)

Traverses the requirement tree, looking for *kernel\_virtual\_offset* values that may need setting and sets it based on the previously identified *valid\_kernels*.

**Parameters**

- **context** (*ContextInterface*) – Context on which to operate and provide the kernel virtual offset
- **valid\_kernels** (*Dict[str, Tuple[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]`

**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 strs 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`)

**scan** (*ctx*, *layer\_name*, *page\_size*, *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]`

## 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*

**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\_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** `unsatisfied(context, config_path)`

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be 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\_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\_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.DtbTest32bit object>*, *<volatility.fra*

The default tests to run when searching for DTBs

**thread\_safe** = True

**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 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.automagic.windows.DtbTest32bit 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]`

**class** `WintelStacker`

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

**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 = 0`

## 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]]]`

**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 `builtins.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]]]`

**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 `builtins.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]]]*

**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]]]*

**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]]]`

**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 `builtins.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]]]`

**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]]]*

**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]]]*

**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=None, optional=False, plugin=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]]]`

**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]]]`

**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 `builtins.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]]]*

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

Class maintaining the limitations on what sort of translation layers are acceptable.

Constructs a Translation Layer Requirement.

The configuration option’s value will be the name of the layer once it exists in the store

**Parameters**

- **name** (`str`) – Name of the configuration requirement
- **description** (`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[~T]]`) – A list of valid operating systems which can satisfy this requirement
- **architectures** (`Optional[List[~T]]`) – 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]]]`

**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]]]`

**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 `builtins.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

## 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

**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 = '3.0.0\_alpha1'**

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/foobarv3/checkouts/latest'**

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

**class Parallelism**

Bases: `enum.IntEnum`

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

**Multiprocessing = 2**

**Off = 0**

**Threading = 1**

**ProgressCallback = typing.Union[typing.Callable[[float, str], NoneType], NoneType]**

Type information for ProgressCallback objects

**SYMBOL\_BASEPATHS = ['/home/docs/checkouts/readthedocs.org/user\_builds/foobarv3/checkouts/latest'**

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 occupies 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`

**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

**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*[+CT\_co]

**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]]]

**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]]]`

**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]]]*

**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*[~KT, ~VT]

**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]]]*

**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]]]*

**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 *builtins.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

**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\_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[~KT, +VT_co]`

**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: *object*

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**

**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, mapped\_offset, 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, 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[~KT, +VT_co]`

**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** (`self, 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*)

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

Contains common information useful/pertinent only to an individual object (like an instance)

This typically contains information such as the layer the object belongs to, the offset where it was constructed, and if it is a subordinate object, its parent.

This is primarily used to reduce the number of parameters passed to object constructors and keep them all together in a single place. These values are based on the *ReadOnlyMapping* class, to prevent their modification.

Constructs a container for basic information about an object.

#### Parameters

- **layer\_name** (*str*) – Layer from which the data for the object will be read
- **offset** (*int*) – Offset within the layer at which the data for the object will be read

- **member\_name** (`Optional[str]`) – If the object was accessed as a member of a parent object, this was the name used to access it
- **parent** (`Optional[ObjectInterface]`) – If the object was accessed as a member of a parent object, this is the parent object
- **native\_layer\_name** (`Optional[str]`) – If this object references other objects (such as a pointer), what layer those objects live in

**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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**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*

**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 as least the basic information that each object will require before it is instantiated (so *offset* and *parent* are explicitly not recorded here). This dictionary can be updated after construction, but any changes made after that point will *not* be cloned. This is so that templates such as those for string objects may contain different length limits, without affecting all other strings using the same template from a `SymbolTable`, constructed at resolution time and then cached.

Stores the keyword arguments for later object creation.

**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 FileConsumerInterface**

Bases: `object`

Class for consuming files potentially produced by plugins.

We use the producer/consumer model to ensure we can avoid running out of memory by storing every file produced. The downside is, we can't provide much feedback to the producer about what happened to their file (other than exceptions).

**consume\_file** (*file*)

Consumes a file as passed back to a UI by a plugin.

**Parameters** **file** (*FileInterface*) – A FileInterface object with the data to write to a file

**Return type** None

**class FileInterface** (*filename, data=None*)

Bases: *object*

Class for storing Files in the plugin as a means to output a file or files when necessary.

**Parameters**

- **filename** (*str*) – The requested name of the filename for the data
- **data** (*Optional[bytes]*) – The data to be stored in a file

**class PluginInterface** (*context, config\_path, progress\_callback=None*)

Bases: *volatility.framework.interfaces.configuration.ConfigurableInterface*

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`

**produce\_file** (`filedata`)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**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\_file\_consumer** (`consumer`)

Sets the file consumer 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** ()

Return number of occurrences of value.

**index** ()

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[+T_co, -T_contra, +V_co]`) – 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 '...`

**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*)

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** `None`

**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 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** `Iterable[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[Dict[str, Type[ObjectInterface]]]`) – A dictionary of types and classes that should be instantiated instead of Struct to construct them

**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[Dict[str, Type[ObjectInterface]]]`) – A dictionary of types and classes that should be instantiated instead of `Struct` to construct them

**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`

**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. 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** – 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[Dict[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*

**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.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`

**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`

**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`

### 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, Union[str, bytes]], None, None]`

## Submodules

### `volatility.framework.layers.crash` module

**exception** `WindowsCrashDump32FormatException` (*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** `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`

**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]`

**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, mapped\_offset, length, layer) mappings.

**Return type** `Iterable[Tuple[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[~KT, +VT_co]`

**property minimum\_address**

Returns the minimum valid address of the space.

**Return type** `int`

**property name**

Returns the layer name.

**Return type** `str`

**priority** = 23

**provides** = {'type': 'physical'}

**read** (*self*, *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 WindowsCrashDump32Stacker**

Bases: *volatility.framework.interfaces.automatic.StackerLayerInterface*

**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

**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, mapped\_offset, length, layer) mappings.

This allows translation layers to provide maps of contiguous regions in one layer

**Return type** `Iterable[Tuple[int, int, int, str]]`

**maximum\_address** = 4294967295

**property metadata**

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

**Return type** `Mapping[~KT, +VT_co]`

**minimum\_address** = 0

**property name**

Returns the layer name.

**Return type** `str`

**page\_size** = 4096

**priority** = 40

**read** (*self, 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, mapped\_offset, length, layer) mappings.

This allows translation layers to provide maps of contiguous regions in one layer

**Return type** `Iterable[Tuple[int, int, int, str]]`

**maximum\_address = 281474976710655**

**property metadata**

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

**Return type** `Mapping[~KT, +VT_co]`

**minimum\_address = 0**



**property name**

Returns the layer name.

**Return type** `str`

**page\_size** = 4096

**priority** = 30

**read** (*self*, *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, mapped\_offset, length, layer) mappings.

This allows translation layers to provide maps of contiguous regions in one layer

**Return type** `Iterable[Tuple[int, int, int, str]]`

**maximum\_address** = 4294967295

**property metadata**

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

**Return type** `Mapping[~KT, +VT_co]`

**minimum\_address** = 0

**property name**

Returns the layer name.

**Return type** `str`

**page\_size** = 4096

**priority** = 35

**read** (*self, 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, mapped\_offset, length, layer) mappings.

This allows translation layers to provide maps of contiguous regions in one layer

**Return type** *Iterable[Tuple[int, int, int, str]]*

**maximum\_address** = 4294967295

**property metadata**

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

**Return type** *Mapping[~KT, +VT\_co]*

**minimum\_address** = 0

**property name**

Returns the layer name.

**Return type** *str*

**page\_size** = 4096

**priority** = 40

**read** (*self, 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, mapped\_offset, length, layer) mappings.

This allows translation layers to provide maps of contiguous regions in one layer

**Return type** `Iterable[Tuple[int, int, int, str]]`

**maximum\_address** = 281474976710655

**property metadata**

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

**Return type** `Mapping[~KT, +VT_co]`

**minimum\_address** = 0

**property name**

Returns the layer name.

**Return type** `str`

**page\_size** = 4096

**priority** = 30

**read(self, 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, mapped\_offset, length, layer) mappings.

This allows translation layers to provide maps of contiguous regions in one layer

**Return type** `Iterable[Tuple[int, int, int, str]]`

**maximum\_address = 4294967295**

**property metadata**

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

**Return type** `Mapping[~KT, +VT_co]`

**minimum\_address** = 0

**property name**

Returns the layer name.

**Return type** `str`

**page\_size** = 4096

**priority** = 35

**read** (*self*, *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, mapped\_offset, length, layer) mappings.

This allows translation layers to provide maps of contiguous regions in one layer

**Return type** `Iterable[Tuple[int, int, int, str]]`

**maximum\_address** = 4294967295

**property metadata**

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

**Return type** `Mapping[~KT, +VT_co]`

**minimum\_address** = 0

**property name**

Returns the layer name.

**Return type** `str`

**page\_size** = 4096

**priority** = 40

**read** (*self, 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, mapped\_offset, length, layer) mappings.

**Return type** `Iterable[Tuple[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[~KT, +VT_co]`

**property minimum\_address**

Returns the minimum valid address of the space.

**Return type** `int`

**property name**

Returns the layer name.

**Return type** `str`

**priority** = 21

**read** (*self*, *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*

**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

## 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, mapped\_offset, 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, 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[~KT, +VT\_co]*

**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** (*self*, *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**

**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, mapped\_offset, 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, 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[~KT, +VT_co]`

**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** (*self*, *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, mapped\_offset, 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, 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[~KT, +VT_co]`**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** (*self*, *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[~KT, +VT_co]`

**property minimum\_address**

Returns the smallest available address in the space.

**Return type** `int`

**property name**

Returns the layer name.

**Return type** `str`

**priority = 10****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*[~KT, +VT\_co]

**property minimum\_address**

Returns the smallest available address in the space.

**Return type** *int*

**property name**

Returns the layer name.

**Return type** *str*

**priority** = 20

**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.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*, *mapped\_offset*, *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, 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[~KT, +VT_co]`

**property minimum\_address**

Returns the minimum valid address of the space.

**Return type** `int`

**property name**

Returns the layer name.

**Return type** `str`

**read** (*self*, *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.

**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

**open** (*url, mode='rb'*)

Returns a file-like object for a particular URL opened in mode.

## volatility.framework.layers.segmented module

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

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

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, mapped\_offset, length, layer) mappings.

**Return type** *Iterable[Tuple[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[~KT, +VT\_co]*

**property minimum\_address**

Returns the minimum valid address of the space.

**Return type** *int*

**property name**

Returns the layer name.

**Return type** *str*

**read** (*self, 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

**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, mapped\_offset, length, layer) mappings.

**Return type** `Iterable[Tuple[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[~KT, +VT_co]`

**property minimum\_address**

Returns the minimum valid address of the space.

**Return type** `int`

**property name**

Returns the layer name.

**Return type** `str`

**priority** = 22

**read** (*self*, *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.automatic.StackerLayerInterface*

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

Attempt to stack this based on the starting information.

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

**stack\_order** = 0

## 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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called *member\_name*.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**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`

**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.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** ()

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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**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*

**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** ()

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.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()**

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()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**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*

**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()**

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.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`

**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()**

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()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**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*

**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()**

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()**

Strip leading bytes contained in the argument.

If the argument is omitted or None, strip leading ASCII whitespace.

**static maketrans()**

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()**

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()**

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** ()

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** ()

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** ()

Strip trailing bytes contained in the argument.

If the argument is omitted or None, strip trailing ASCII whitespace.

**split** ()

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** ()

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** ()

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()**

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.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** ()

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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**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*

**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** ()

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.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()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member(member\_name)**

Returns whether the object would contain a member called member\_name.

**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()**

Return number of occurrences of value.

**index()**

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.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** ()

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()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**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*

**imag**

the imaginary part of a complex number

**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()**

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.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()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**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*

**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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`**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`**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.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()**

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()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**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*

**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()**

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.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** ()

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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member** (*member\_name*)

Returns whether the dereferenced type has this member.

**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** ()

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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**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`

**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.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`

**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** ()

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

**count** (*sub* [, *start* [, *end* ]]) → int

Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

**encode** ()

Encode the string using the codec registered for encoding.

**encoding** The encoding in which to encode the string.

**errors** The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a `UnicodeEncodeError`. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeEncodeErrors`.

**endswith** (*suffix* [, *start* [, *end* ]]) → bool

Return True if *S* ends with the specified suffix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *suffix* can also be a tuple of strings to try.

**expandtabs** ()

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

**find** (*sub* [, *start* [, *end* ]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

**format** (*\*args*, *\*\*kwargs*) → str

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

**format\_map** (*mapping*) → str

Return a formatted version of *S*, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

**get\_symbol\_table** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**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`

**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()**

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()**

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()**

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

**static maketrans()**

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

**partition()**

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()**

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** ()

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

**rpartition** ()

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** ()

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** ()

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not `None`, remove characters in *chars* instead.

**split** ()

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** ()

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** ()

Return a copy of the string with leading and trailing whitespace remove.

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()**

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()**

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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called *member\_name*.

**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.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** ()  
Returns the symbol table for this particular object.  
Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (`member_name`)  
Returns whether the object would contain a member called `member_name`.

**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.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.

**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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**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`

**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.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.

## 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*

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer(consumer)**

Sets the file consumer 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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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*

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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`

**produce\_file** (`filedata`)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects `self.validate` to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (`consumer`)

Sets the file consumer 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** `Lsmod` (*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

**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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.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*

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer(consumer)**

Sets the file consumer 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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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`

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run()**

Executes the functionality of the code.

---

**Note:** This method expects `self.validate` to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer(consumer)**

Sets the file consumer 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`

**produce\_file** (`filedata`)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects `self.validate` to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (`consumer`)

Sets the file consumer 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`

**produce\_file** (`filedata`)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects `self.validate` to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (`consumer`)

Sets the file consumer 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 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.

## 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`

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run()**

Executes the functionality of the code.

---

**Note:** This method expects `self.validate` to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer(consumer)**

Sets the file consumer 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`**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** None

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer**(*consumer*)

Sets the file consumer 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`

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run()**

Executes the functionality of the code.

---

**Note:** This method expects `self.validate` to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer(consumer)**

Sets the file consumer 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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** None

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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*

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer(consumer)**

Sets the file consumer 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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.lsof module

**class** `lsof(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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.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 make\_subconfig** (*context*, *base\_config\_path*, *\*\*kwargs*)

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

**Parameters**

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

**Returns** The newly generated full configuration path

**Return type** *str*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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*

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Return type** *TreeGrid*

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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_requirements` ()

Returns a list of Requirement objects for this plugin.

**classmethod** `list_tasks` (*context*, *layer\_name*, *darwin\_symbols*, *filter\_func=<function PsList.<lambda>>>*)

Lists all the processes 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
- **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** `make_subconfig` (*context*, *base\_config\_path*, *\*\*kwargs*)

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

**Parameters**

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

**Returns** The newly generated full configuration path

**Return type** *str*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.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`

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.tasks module

**class Tasks** (*context, config\_path, progress\_callback=None*)

Bases: *volatility.plugins.mac.pslist.PsList*

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\_requirements** ()

Returns a list of Requirement objects for this plugin.

**classmethod list\_tasks** (*context*, *layer\_name*, *darwin\_symbols*, *filter\_func=<function Tasks.<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
- **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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer**(*consumer*)

Sets the file consumer 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.trustedbsd 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`

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run()**

Executes the functionality of the code.

---

**Note:** This method expects `self.validate` to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer(consumer)**

Sets the file consumer 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.

## 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.

## Submodules

### volatility.plugins.windows.registry.certificates module

```
class Certificates (context, config_path, progress_callback=None)
```

Bases: *volatility.framework.interfaces.plugins.PluginInterface*

Lists the certificates in the registry's Certificate Store.

#### 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`

**parse\_data(data)****produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run()**

Executes the functionality of the code.

---

**Note:** This method expects `self.validate` to have been called to ensure all necessary options have been provided

---

**Return type** `TreeGrid`

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer(consumer)**

Sets the file consumer 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.registry.hivelist module

**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
- **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`

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Return type** `TreeGrid`

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**classmethod scan\_hives** (*context, layer\_name, symbol\_table*)

Scans for hives 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 Hive objects as found from the *layer\_name* layer based on Hive pool signatures

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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*

**parse\_userassist\_data(reg\_val)**

Reads the raw data of a \_CM\_KEY\_VALUE and returns a dict of userassist fields.

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.statistics module

**class Statistics** (*context, config\_path, progress\_callback=None*)

Bases: *volatility.framework.interfaces.plugins.PluginInterface*

#### 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`

**produce\_file** (`filedata`)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects `self.validate` to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (`consumer`)

Sets the file consumer 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.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\_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`**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** None

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer**(*consumer*)

Sets the file consumer 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.dlldump module

**class DllDump**(*context, config\_path, progress\_callback=None*)

Bases: *volatility.framework.interfaces.plugins.PluginInterface*

Dumps process memory ranges as DLLs.

### 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`

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run()**

Executes the functionality of the code.

---

**Note:** This method expects `self.validate` to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer(consumer)**

Sets the file consumer 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.dlllist module

```
class DllList (context, config_path, progress_callback=None)
```

Bases: *volatility.framework.interfaces.plugins.PluginInterface*

Lists the loaded modules in a particular windows memory image.

#### Parameters

- **context** (*ContextInterface*) – The context that the plugin will operate within
- **config\_path** (*str*) – The path to configuration data within the context configuration data
- **progress\_callback** (*Optional[Callable[[float, str], None]]*) – A callable that can provide feedback at progress points

```
build_configuration()
```

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

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

**Return type** *HierarchicalDict*

```
property config
```

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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** None

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.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`

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run()**

Executes the functionality of the code.

---

**Note:** This method expects `self.validate` to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer(consumer)**

Sets the file consumer 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`

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** None

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**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\_file\_consumer** (*consumer*)

Sets the file consumer 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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**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\_file\_consumer** (*consumer*)

Sets the file consumer 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.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`

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** None

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer**(*consumer*)

Sets the file consumer 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.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\_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`

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.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, 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
- **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`

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects `self.validate` to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.moddump module

**class** `ModDump` (*context, config\_path, progress\_callback=None*)

Bases: `volatility.framework.interfaces.plugins.PluginInterface`

Dumps 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** **make\_subconfig** (*context, base\_config\_path, \*\*kwargs*)

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

**Parameters**

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

**Returns** The newly generated full configuration path

**Return type** *str*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.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** `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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**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\_file\_consumer** (*consumer*)

Sets the file consumer 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.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 get\_requirements** ()

Returns a list of Requirement objects for this plugin.

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

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.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`

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run()**

Executes the functionality of the code.

---

**Note:** This method expects `self.validate` to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**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\_file\_consumer** (*consumer*)

Sets the file consumer 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.poolscanner module****class PoolConstraint** (*tag, type\_name, object\_type=None, page\_type=None, size=None, index=None, alignment=1*)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****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\_requirements()**

Returns a list of Requirement objects for this plugin.

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

**is\_windows\_10** (*symbol\_table: str*) → bool

**Parameters**

- **context** (*ContextInterface*) – The context that contains the symbol table named *symbol\_table*
- **symbol\_table** (*str*) – Name of the symbol table within the context to distinguish the version of

**Return type** `bool`

**Returns** True if the symbol table is of the required version

**is\_windows\_7** (*symbol\_table: str*) → bool

**Parameters**

- **context** (*ContextInterface*) – The context that contains the symbol table named *symbol\_table*
- **symbol\_table** (*str*) – Name of the symbol table within the context to distinguish the version of

**Return type** `bool`

**Returns** True if the symbol table is of the required version

**is\_windows\_8\_or\_later** (*symbol\_table: str*) → bool

**Parameters**

- **context** (*ContextInterface*) – The context that contains the symbol table named *symbol\_table*
- **symbol\_table** (*str*) – Name of the symbol table within the context to distinguish the version of

**Return type** `bool`

**Returns** True if the symbol table is of the required version

**classmethod make\_subconfig** (*context, base\_config\_path, \*\*kwargs*)

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

**Parameters**

- **context** (*ContextInterface*) – The context in which to store the new configuration
- **base\_config\_path** (*str*) – The base configuration path on which to build the 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 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

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Return type** *TreeGrid*

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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**

Bases: *enum.IntEnum*

Class to maintain the different possible PoolTypes The values must be integer powers of 2.

**FREE** = 4

**NONPAGED** = 2

PAGED = 1

**os\_distinguisher** (*version\_check, fallback\_checks*)

Distinguishes a symbol table as being above a particular version or point.

This will primarily check the version metadata first and foremost. If that metadata isn't available then each item in the *fallback\_checks* is tested. If *invert* is specified then the result will be true if the version is less than that specified, or in the case of *fallback*, if any of the fallback checks is successful.

**A fallback check is made up of:**

- a symbol or type name
- a member name (implying that the value before was a type name)
- whether that symbol, type or member must be present or absent for the symbol table to be more above the required point

---

**Note:** Specifying that a member must not be present includes the whole type not being present too (ie, either will pass the test)

---

#### Parameters

- **version\_check** (*Callable*[[*Tuple*[*int*, ...]], *bool*]) – Function that takes a 4-tuple version and returns whether whether the provided version is above a particular point
- **fallback\_checks** (*List*[*Tuple*[*str*, *Optional*[*str*], *bool*]]) – A list of symbol/types/members of types, and whether they must be present to be above the required point

**Return type** *Callable*[[*ContextInterface*, *str*], *bool*]

**Returns** A function that takes a context and a symbol table name and determines whether that symbol table passes the distinguishing checks

### volatility.plugins.windows.procdump module

**class ProcDump** (*context, config\_path, progress\_callback=None*)

Bases: *volatility.framework.interfaces.plugins.PluginInterface*

Dumps process executable images.

#### 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*

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer(consumer)**

Sets the file consumer 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.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`

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.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\_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`

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**classmethod scan\_processes(context, layer\_name, symbol\_table)**

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\_file\_consumer(consumer)**

Sets the file consumer 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.pstree module

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

Bases: `volatility.plugins.windows.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

**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

**find\_level** (*pid*)

Finds how deep the pid is in the processes list.

**Return type** `None`

**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`

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer**(*consumer*)

Sets the file consumer 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.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*

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Return type** *TreeGrid*

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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`

**produce\_file** (`filedata`)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects `self.validate` to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (`consumer`)

Sets the file consumer 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.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`

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**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\_file\_consumer** (*consumer*)

Sets the file consumer 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.vaddump module

**class VadDump** (*context, config\_path, progress\_callback=None*)

Bases: *volatility.framework.interfaces.plugins.PluginInterface*

Dumps process memory ranges.

**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*

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run()**

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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

**build\_configuration** ()

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

Ensures that if the class has been created, it 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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**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\_file\_consumer** (*consumer*)

Sets the file consumer 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.vadyscan 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`

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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(*args, **kwargs)`

Bases: `volatility.framework.interfaces.plugins.PluginInterface`

Lists virtual mapped sections.

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 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`

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** None

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**classmethod scannable\_sections** (*module*)

**Return type** Generator[Tuple[int, int], None, None]

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.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`

**produce\_file(filedata)**

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** `None`

**run()**

Executes the functionality of the code.

---

**Note:** This method expects `self.validate` to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer(consumer)**

Sets the file consumer 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.layerwriter module

**class** `LayerWriter` (*context, config\_path, progress\_callback=None*)

Bases: `volatility.framework.interfaces.plugins.PluginInterface`

Runs the automagics and lists out the generated layers if no layer name is specified, otherwise writes out the named layer.

### 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

**default\_output\_name** = 'output.raw'

**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*

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** *None*

**run** ()

Executes the functionality of the code.

---

**Note:** This method expects *self.validate* to have been called to ensure all necessary options have been provided

---

**Returns** A TreeGrid object that can then be passed to a Renderer.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.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**

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[+CT_co]]`

**classmethod make\_subconfig** (context, base\_config\_path, \*\*kwargs)

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

**Parameters**

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

**Returns** The newly generated full configuration path

**Return type** `str`

**produce\_file** (*filedata*)

Adds a file to the plugin's file store and returns the chosen filename for the file.

**Return type** None

**run** ()

Isolate each plugin and run it.

**set\_file\_consumer** (*consumer*)

Sets the file consumer 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.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.

**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 '...

**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*)

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** `None`

**property populated**

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

**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** `Iterable[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()**

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()**

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'
```

(continues on next page)

(continued from previous page)

```
>>> (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()**

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()**

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** ()

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** ()

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** ()

Strip leading bytes contained in the argument.

If the argument is omitted or None, strip leading ASCII whitespace.

**static maketrans** ()

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** ()

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** ()

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** ()

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** ()

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** ()

Strip trailing bytes contained in the argument.

If the argument is omitted or None, strip trailing ASCII whitespace.

**split** ()

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** ()

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** ()

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** ()

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*

**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**

Bases: `enum.Enum`

An enumeration.

**ENUM** = 3

**SYMBOL** = 2

**TYPE** = 1

**mask\_symbol\_table** (*symbol\_table*, *address\_mask*=0, *table\_aslr\_shift*=0)

Alters a symbol table, such that all symbols returned have their address masked by the address mask.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called *member\_name*.

**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(context, config_path, name, isf_url)`

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** (`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** – 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

**build\_configuration()**

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

Ensures that if the class has been created, it 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**(*context*, *config\_path*, *sub\_path*, *filename*, *native\_types=None*, *table\_mapping=None*, *class\_types=None*)

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

**Return type** *str*

**Returns** the name of the added symbol table

**del\_type\_class**(*\*args*, *\*\*kwargs*)

**property enumerations**

**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)

**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)

**get\_type\_class** (\*args, \*\*kwargs)

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

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

#### Parameters

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base\_config\_path** (`str`) – The base configuration path on which to build the 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)

**property** `symbols`

**property** `types`

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

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be 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.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.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()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member**(*member\_name*)

Returns whether the object would contain a member called *member\_name*.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member** (*member\_name*)

Returns whether the object would contain a member called *member\_name*.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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_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.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 ()
    Returns the symbol table for this particular object.

    Returns none if the symbol table cannot be identified.

    Return type SymbolTableInterface

has_member (member_name)
    Returns whether the object would contain a member called member_name.

    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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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\_symbol\_table()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member(member\_name)**

Returns whether the object would contain a member called member\_name.

**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 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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member** (*member\_name*)

Returns whether the object would contain a member called *member\_name*.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**get\_vfsmnt** ()

**Return type** *ObjectInterface*

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (`member_name`)

Returns whether the object would contain a member called member\_name.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called *member\_name*.

**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.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()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**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*

**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.

## 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

**build\_configuration()**

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

Ensures that if the class has been created, it 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**(*context*, *config\_path*, *sub\_path*, *filename*, *native\_types=None*, *table\_mapping=None*, *class\_types=None*)

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

**Return type** *str*

**Returns** the name of the added symbol table

**del\_type\_class**(*\*args*, *\*\*kwargs*)

**property enumerations**

**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)

**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)

**get\_type\_class** (\*args, \*\*kwargs)

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

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

**Parameters**

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base\_config\_path** (`str`) – The base configuration path on which to build the 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)

**property** `symbols`

**property** `types`

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

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be 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** (*context, config\_path, name, isf\_url*)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** (*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** – 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

**build\_configuration** ()

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

Ensures that if the class has been created, it 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** (*context, config\_path, sub\_path, filename, native\_types=None, table\_mapping=None, class\_types=None*)

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

**Return type** *str*

**Returns** the name of the added symbol table

**del\_type\_class** (*\*args, \*\*kwargs*)

**property enumerations**

**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*)

**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*)

**get\_type\_class** (*\*args, \*\*kwargs*)

**classmethod make\_subconfig** (*context, base\_config\_path, \*\*kwargs*)

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

**Parameters**

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

- **base\_config\_path** (*str*) – The base configuration path on which to build the 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)

**property symbols**

**property types**

**classmethod unsatisfied** (context, config\_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be 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.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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**get\_tcp\_state** ()**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**get\_task()**

**has\_member**(*member\_name*)

Returns whether the object would contain a member called *member\_name*.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member** (*member\_name*)

Returns whether the object would contain a member called *member\_name*.

**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*)

**Return type** *Iterable[ObjectInterface]*

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (`member_name`)

Returns whether the object would contain a member called `member_name`.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**get\_vnode** (*context*, *config\_prefix*)

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called *member\_name*.

**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.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** ()  
Returns the symbol table for this particular object.  
Returns none if the symbol table cannot be identified.  
**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)  
Returns whether the object would contain a member called member\_name.  
**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` (*context, config\_path, name, isf\_url*)  
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** (`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** – 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

**build\_configuration** ()

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

Ensures that if the class has been created, it 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** (*context*, *config\_path*, *sub\_path*, *filename*, *native\_types=None*, *table\_mapping=None*, *class\_types=None*)

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

**Return type** *str*

**Returns** the name of the added symbol table

**del\_type\_class** (\*args, \*\*kwargs)

**property enumerations**

**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)

**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)

**get\_type\_class** (\*args, \*\*kwargs)

**classmethod make\_subconfig** (context, base\_config\_path, \*\*kwargs)

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

**Parameters**

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base\_config\_path** (`str`) – The base configuration path on which to build the 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)

**property symbols**

**property types****classmethod** `unsatisfied` (*context*, *config\_path*)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be 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** `DEVICE_OBJECT` (*context*, *type\_name*, *object\_info*, *size*, *members*)Bases: `volatility.framework.objects.StructType`, `volatility.framework.symbols.windows.extensions.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.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\_symbol\_table** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**Return type** *bool*

**member** (*attr='member'*)

Specifically named method for retrieving members.

**Return type** *object*

**object\_header** ()

**Return type** *OBJECT\_HEADER*

**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.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.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\_symbol\_table** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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`

**object\_header** ()

**Return type** `OBJECT_HEADER`

**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.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.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`

**get\_create\_time** ()

**get\_exit\_time** ()

**get\_handle\_count** ()

**get\_is\_wow64** ()

**get\_session\_id** ()

**get\_symbol\_table** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**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`

**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[int]`

**member** (*attr*=*'member'*)

Specifically named method for retrieving members.

**Return type** `object`

**object\_header** ()

**Return type** `OBJECT_HEADER`

**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.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** `()`

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** `(member_name)`

Returns whether the object would contain a member called member\_name.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called *member\_name*.

**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 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\_symbol\_table** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**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*

**object\_header** ()

**Return type** *OBJECT\_HEADER*

**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** **FILE\_OBJECT** (*context, type\_name, object\_info, size, members*)

Bases: *volatility.framework.objects.StructType, volatility.framework.symbols.windows.extensions.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.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*

**file\_name\_with\_device()**

**Return type** *Union[str, BaseAbsentValue]*

**get\_symbol\_table()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member(member\_name)**

Returns whether the object would contain a member called member\_name.

**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*

**object\_header()**

**Return type** *OBJECT\_HEADER*

**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.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.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\_symbol\_table** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (`member_name`)

Returns whether the object would contain a member called member\_name.

**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`

**object\_header** ()

**Return type** `OBJECT_HEADER`

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**get\_time** ()

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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** ()

Returns the symbol table for this particular object.



Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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*, *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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**get\_tag**

**has\_member** (*member\_name*)

Returns whether the object would contain a member called *member\_name*.

**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.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, 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()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**get\_tag**

**has\_member(member\_name)**

Returns whether the object would contain a member called member\_name.

**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\_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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

Return type *SymbolTableInterface*

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

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** `OBJECT_SYMBOLIC_LINK` (*context, type\_name, object\_info, size, members*)

Bases: `volatility.framework.objects.StructType`, `volatility.framework.symbols.windows.extensions.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.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\_symbol\_table()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member(member\_name)**

Returns whether the object would contain a member called member\_name.

**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*

**object\_header()**

**Return type** *OBJECT\_HEADER*

**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.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, type\_map, use\_top\_down, native\_layer\_name=None, object\_type=None, cookie=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** (`Optional[str]`) – the object type (executive kernel objects only)

**Return type** `Optional[ObjectInterface]`

**Returns**

**get\_symbol\_table** ()  
Returns the symbol table for this particular object.  
Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)  
Returns whether the object would contain a member called member\_name.

**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** **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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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()**  
Returns the symbol table for this particular object.  
Returns none if the symbol table cannot be identified.  
**Return type** *SymbolTableInterface*

**has\_member(member\_name)**  
Returns whether the object would contain a member called member\_name.  
**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.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.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** `()`

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** `(member_name)`

Returns whether the object would contain a member called `member_name`.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member** (*member\_name*)

Returns whether the object would contain a member called *member\_name*.

**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.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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called *member\_name*.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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()**

Since this is just a casting convenience, it can be a property.

**Return type** *ObjectInterface*

**get\_subkeys()**

Returns a list of the key nodes.

**Return type** *Iterable[ObjectInterface]*

**get\_symbol\_table()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**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*

**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.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** ()

Since this is just a casting convenience, it can be a property.

**Return type** `Union[str, bytes]`

**get\_name** ()

Since this is just a casting convenience, it can be a property.

**Return type** `ObjectInterface`

**get\_symbol\_table** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** `SymbolTableInterface`

**has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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

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

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

get = <bound method RegValueTypes.get of <enum 'RegValueTypes'>>

**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.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** ()

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface***has\_member** (*member\_name*)

Returns whether the object would contain a member called member\_name.

**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.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()**

Returns the symbol table for this particular object.

Returns none if the symbol table cannot be identified.

**Return type** *SymbolTableInterface*

**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*

**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, 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[~T], 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\_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`

**get\_report\_hook** (*progress\_callback, url*)

Returns a report hook that converts output into a progress\_callback.

**retrieive\_pdb** (*guid, file\_name, progress\_callback=None*)

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

## 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. 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** – 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*

**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*)

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[Dict[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*

**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*)

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

**Return type** `str`

**Returns** the name of the added symbol table

**del\_type\_class** (`*args, **kwargs`)

**property enumerations**

**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`)

**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`)

**get\_type\_class** (`*args, **kwargs`)

**classmethod make\_subconfig** (`context, base_config_path, **kwargs`)

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

**Parameters**

- **context** (`ContextInterface`) – The context in which to store the new configuration
- **base\_config\_path** (`str`) – The base configuration path on which to build the 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)

**property symbols**

**property types**

**classmethod unsatisfied** (context, config\_path)

Returns a list of the names of all unsatisfied requirements.

Since a satisfied set of requirements will return [], it can be 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. 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** – 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*

**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. 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** – 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*

**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. 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** – 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`

**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. 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** – 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*

**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. 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** – 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`

**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. 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** – 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*

**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. 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** – 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`

**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)

## 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

**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 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**

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.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.

**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, 21  
volatility.cli, 21  
volatility.cli.text\_renderer, 33  
volatility.cli.volshell, 23  
volatility.cli.volshell.generic, 23  
volatility.cli.volshell.linux, 26  
volatility.cli.volshell.mac, 28  
volatility.cli.volshell.windows, 30  
volatility.framework, 34  
volatility.framework.automagic, 35  
volatility.framework.automagic.construct\_layers, 36  
volatility.framework.automagic.linux, 37  
volatility.framework.automagic.mac, 41  
volatility.framework.automagic.pdbscan, 44  
volatility.framework.automagic.stacker, 48  
volatility.framework.automagic.symbol\_cache, 50  
volatility.framework.automagic.symbol\_finder, 52  
volatility.framework.automagic.windows, 54  
volatility.framework.configuration, 59  
volatility.framework.configuration.requirements, 60  
volatility.framework.constants, 79  
volatility.framework.constants.linux, 80  
volatility.framework.constants.windows, 81  
volatility.framework.contexts, 81  
volatility.framework.exceptions, 424  
volatility.framework.interfaces, 86  
volatility.framework.interfaces.automagic, 86  
volatility.framework.interfaces.configuration, 89  
volatility.framework.interfaces.context, 99  
volatility.framework.interfaces.layers, 102  
volatility.framework.interfaces.objects, 109  
volatility.framework.interfaces.plugins, 112  
volatility.framework.interfaces.renderers, 114  
volatility.framework.interfaces.symbols, 118  
volatility.framework.layers, 125  
volatility.framework.layers.crash, 126  
volatility.framework.layers.intel, 129  
volatility.framework.layers.lime, 145  
volatility.framework.layers.linear, 148  
volatility.framework.layers.msf, 150  
volatility.framework.layers.physical, 155  
volatility.framework.layers.registry, 160  
volatility.framework.layers.resources, 163  
volatility.framework.layers.scanners, 125  
volatility.framework.layers.scanners.multiregexp, 125  
volatility.framework.layers.segmented, 163  
volatility.framework.layers.vmware, 166  
volatility.framework.objects, 168  
volatility.framework.objects.templates, 202  
volatility.framework.objects.utility, 203  
volatility.framework.renderers, 307  
volatility.framework.renderers.conversion, 310  
volatility.framework.renderers.format\_hints, 310  
volatility.framework.symbols, 316  
volatility.framework.symbols.generic,

318  
volatility.framework.symbols.intermed,  
399  
volatility.framework.symbols.linux, 319  
volatility.framework.symbols.linux.bash,  
339  
volatility.framework.symbols.linux.extensions,  
322  
volatility.framework.symbols.linux.extensions.bash,  
338  
volatility.framework.symbols.mac, 342  
volatility.framework.symbols.mac.extensions,  
344  
volatility.framework.symbols.metadata,  
421  
volatility.framework.symbols.native, 422  
volatility.framework.symbols.windows,  
357  
volatility.framework.symbols.windows.extensions,  
360  
volatility.framework.symbols.windows.extensions.dbg,  
382  
volatility.framework.symbols.windows.extensions.info,  
383  
volatility.framework.symbols.windows.extensions.registry,  
386  
volatility.framework.symbols.windows.extensions.services,  
394  
volatility.framework.symbols.windows.pdbcache,  
397  
volatility.framework.symbols.wrappers,  
423  
volatility.plugins, 204  
volatility.plugins.configwriter, 302  
volatility.plugins.layerwriter, 304  
volatility.plugins.linux, 204  
volatility.plugins.linux.bash, 204  
volatility.plugins.linux.check\_afinfo,  
206  
volatility.plugins.linux.check\_syscall,  
207  
volatility.plugins.linux.elfs, 209  
volatility.plugins.linux.lsmod, 210  
volatility.plugins.linux.lsof, 212  
volatility.plugins.linux.malfind, 214  
volatility.plugins.linux.proc, 215  
volatility.plugins.linux.pslist, 217  
volatility.plugins.linux.pstree, 219  
volatility.plugins.mac, 221  
volatility.plugins.mac.bash, 221  
volatility.plugins.mac.check\_syscall,  
223  
volatility.plugins.mac.check\_sysctl, 224  
volatility.plugins.mac.check\_trap\_table,  
226  
volatility.plugins.mac.ifconfig, 227  
volatility.plugins.mac.lsmod, 229  
volatility.plugins.mac.lsof, 230  
volatility.plugins.mac.malfind, 232  
volatility.plugins.mac.netstat, 233  
volatility.plugins.mac.proc\_maps, 235  
volatility.plugins.mac.psaux, 236  
volatility.plugins.mac.pslist, 238  
volatility.plugins.mac.pstree, 240  
volatility.plugins.mac.tasks, 241  
volatility.plugins.mac.trustedbsd, 243  
volatility.plugins.timeliner, 305  
volatility.plugins.windows, 245  
volatility.plugins.windows.cmdline, 256  
volatility.plugins.windows.dlldump, 257  
volatility.plugins.windows.dlllist, 259  
volatility.plugins.windows.driverirp,  
260  
volatility.plugins.windows.driverscan,  
262  
volatility.plugins.windows.filescan, 263  
volatility.plugins.windows.handles, 265  
volatility.plugins.windows.info, 267  
volatility.plugins.windows.malfind, 269  
volatility.plugins.windows.moddump, 271  
volatility.plugins.windows.modscan, 273  
volatility.plugins.windows.modules, 275  
volatility.plugins.windows.mutantscan,  
276  
volatility.plugins.windows.poolscanner,  
278  
volatility.plugins.windows.procdump, 282  
volatility.plugins.windows.pslist, 284  
volatility.plugins.windows.psscan, 286  
volatility.plugins.windows.pstree, 288  
volatility.plugins.windows.registry, 245  
volatility.plugins.windows.registry.certificates,  
245  
volatility.plugins.windows.registry.hivelist,  
247  
volatility.plugins.windows.registry.hivescan,  
249  
volatility.plugins.windows.registry.printkey,  
251  
volatility.plugins.windows.registry.userassist,  
252  
volatility.plugins.windows.ssdt, 290  
volatility.plugins.windows.statistics,  
254  
volatility.plugins.windows.strings, 292  
volatility.plugins.windows.symlinkscan,  
294  
volatility.plugins.windows.vaddump, 295

`volatility.plugins.windows.vadinfo`, [297](#)  
`volatility.plugins.windows.verinfo`, [299](#)  
`volatility.plugins.windows.virtmap`, [301](#)  
`volatility.schemas`, [425](#)  
`volatility.symbols`, [426](#)



## A

- ACCESSED (*TimeLinerType* attribute), 306
- add\_layer() (*Context* method), 81
- add\_layer() (*ContextInterface* method), 99
- add\_layer() (*LayerContainer* method), 105
- add\_parent() (*JarHandler* method), 163
- add\_parser() (*HelpfulSubparserAction* method), 22
- add\_pattern() (*MultiRegexp* method), 125
- add\_process\_layer() (*EPROCESS* method), 363
- add\_process\_layer() (*proc* method), 348
- add\_process\_layer() (*task\_struct* method), 334
- add\_requirement() (*BooleanRequirement* method), 60
- add\_requirement() (*BytesRequirement* method), 61
- add\_requirement() (*ChoiceRequirement* method), 62
- add\_requirement() (*ClassRequirement* method), 89
- add\_requirement() (*ComplexListRequirement* method), 64
- add\_requirement() (*ConfigurableRequirementInterface* method), 91
- add\_requirement() (*ConstructableRequirementInterface* method), 93
- add\_requirement() (*IntRequirement* method), 66
- add\_requirement() (*LayerListRequirement* method), 67
- add\_requirement() (*ListRequirement* method), 69
- add\_requirement() (*MultiRequirement* method), 70
- add\_requirement() (*PluginRequirement* method), 72
- add\_requirement() (*RequirementInterface* method), 96
- add\_requirement() (*SimpleTypeRequirement* method), 98
- add\_requirement() (*StringRequirement* method), 74
- add\_requirement() (*SymbolTableRequirement* method), 75
- add\_requirement() (*TranslationLayerRequirement* method), 77
- add\_requirement() (*URIRequirement* method), 78
- address() (*SymbolInterface* property), 121
- address\_mask() (*BufferDataLayer* property), 155
- address\_mask() (*DataLayerInterface* property), 103
- address\_mask() (*FileLayer* property), 158
- address\_mask() (*Intel* property), 129
- address\_mask() (*Intel32e* property), 131
- address\_mask() (*IntelPAE* property), 133
- address\_mask() (*LimeLayer* property), 145
- address\_mask() (*LinearlyMappedLayer* property), 148
- address\_mask() (*PdbMSFStream* property), 150
- address\_mask() (*PdbMultiStreamFormat* property), 153
- address\_mask() (*RegistryHive* property), 160
- address\_mask() (*SegmentedLayer* property), 164
- address\_mask() (*TranslationLayerInterface* property), 107
- address\_mask() (*VmwareLayer* property), 166
- address\_mask() (*WindowsCrashDump32Layer* property), 126
- address\_mask() (*WindowsIntel* property), 136
- address\_mask() (*WindowsIntel32e* property), 138
- address\_mask() (*WindowsIntelPAE* property), 140
- address\_mask() (*WindowsMixin* property), 142
- AggregateType (class in *volatility.framework.objects*), 168
- AggregateType.VolTemplateProxy (class in *volatility.framework.objects*), 168
- append() (*SymbolSpace* method), 317
- append() (*SymbolSpaceInterface* method), 121
- args (*InvalidAddressException* attribute), 424
- args (*LayerException* attribute), 424
- args (*LimeFormatException* attribute), 145
- args (*PagedInvalidAddressException* attribute), 424
- args (*PluginRequirementException* attribute), 424
- args (*PluginVersionException* attribute), 424
- args (*RegistryFormatException* attribute), 160
- args (*RegistryInvalidIndex* attribute), 163
- args (*SwappedInvalidAddressException* attribute), 425
- args (*SymbolError* attribute), 425
- args (*SymbolSpaceError* attribute), 425
- args (*UnsatisfiedException* attribute), 425
- args (*VolatilityException* attribute), 425

- `args` (*WindowsCrashDump32FormatException attribute*), 126
  - `Array` (class in *volatility.framework.objects*), 169
  - `Array.VolTemplateProxy` (class in *volatility.framework.objects*), 170
  - `array_of_pointers()` (in module *volatility.framework.objects.utility*), 203
  - `array_to_string()` (in module *volatility.framework.objects.utility*), 203
  - `as_integer_ratio()` (*Float method*), 185
  - `ascending` (*ColumnSortKey attribute*), 115, 307
  - `aslr_mask_symbol_table()` (*LinuxUtilities class method*), 40
  - `aslr_mask_symbol_table()` (*MacUtilities class method*), 44
  - `AUTOMAGIC_CONFIG_PATH` (in module *volatility.framework.constants*), 80
  - `AutomagicInterface` (class in *volatility.framework.interfaces.automagic*), 86
  - `available()` (in module *volatility.framework.automagic*), 35
- ## B
- `BANG` (in module *volatility.framework.constants*), 80
  - `banner_cache` (*LinuxSymbolFinder attribute*), 39
  - `banner_cache` (*MacSymbolFinder attribute*), 42
  - `banner_cache` (*SymbolFinder attribute*), 52
  - `banner_config_key` (*LinuxSymbolFinder attribute*), 39
  - `banner_config_key` (*MacSymbolFinder attribute*), 42
  - `banner_config_key` (*SymbolFinder attribute*), 52
  - `banner_path` (*LinuxBannerCache attribute*), 37
  - `banner_path` (*MacBannerCache attribute*), 41
  - `banner_path` (*SymbolBannerCache attribute*), 50
  - `banners()` (*LinuxSymbolFinder property*), 39
  - `banners()` (*MacSymbolFinder property*), 42
  - `banners()` (*SymbolFinder property*), 52
  - `base_types` (*TreeGrid attribute*), 116, 308
  - `BaseAbsentValue` (class in *volatility.framework.interfaces.renderers*), 115
  - `BaseSymbolTableInterface` (class in *volatility.framework.interfaces.symbols*), 118
  - `Bash` (class in *volatility.plugins.linux.bash*), 204
  - `Bash` (class in *volatility.plugins.mac.bash*), 221
  - `BashIntermedSymbols` (class in *volatility.framework.symbols.linux.bash*), 339
  - `Bin` (class in *volatility.framework.renderers.format\_hints*), 310
  - `bit_length()` (*Bin method*), 310
  - `bit_length()` (*BitField method*), 171
  - `bit_length()` (*Boolean method*), 173
  - `bit_length()` (*Char method*), 179
  - `bit_length()` (*Enumeration method*), 183
  - `bit_length()` (*Hex method*), 311
  - `bit_length()` (*Integer method*), 188
  - `bit_length()` (*Pointer method*), 190
  - `BitField` (class in *volatility.framework.objects*), 171
  - `BitField.VolTemplateProxy` (class in *volatility.framework.objects*), 171
  - `bits_per_register` (*Intel attribute*), 129
  - `bits_per_register` (*Intel32e attribute*), 131
  - `bits_per_register` (*IntelPAE attribute*), 134
  - `bits_per_register` (*WindowsIntel attribute*), 136
  - `bits_per_register` (*WindowsIntel32e attribute*), 138
  - `bits_per_register` (*WindowsIntelPAE attribute*), 140
  - `bits_per_register` (*WindowsMixin attribute*), 143
  - `Boolean` (class in *volatility.framework.objects*), 173
  - `Boolean.VolTemplateProxy` (class in *volatility.framework.objects*), 173
  - `BooleanRequirement` (class in *volatility.framework.configuration.requirements*), 60
  - `branch()` (*HierarchicalDict method*), 95
  - `BufferDataLayer` (class in *volatility.framework.layers.physical*), 155
  - `build_configuration()` (*AutomagicInterface method*), 87
  - `build_configuration()` (*Bash method*), 204, 221
  - `build_configuration()` (*BashIntermedSymbols method*), 340
  - `build_configuration()` (*BufferDataLayer method*), 155
  - `build_configuration()` (*Certificates method*), 245
  - `build_configuration()` (*Check\_afinfo method*), 206
  - `build_configuration()` (*Check\_syscall method*), 208, 223, 243
  - `build_configuration()` (*Check\_sysctl method*), 224
  - `build_configuration()` (*Check\_trap\_table method*), 226
  - `build_configuration()` (*CmdLine method*), 256
  - `build_configuration()` (*ComplexListRequirement method*), 64
  - `build_configuration()` (*ConfigurableInterface method*), 90
  - `build_configuration()` (*ConfigurableRequirementInterface method*), 92
  - `build_configuration()` (*ConfigWriter method*), 302
  - `build_configuration()` (*ConstructionMagic method*), 36
  - `build_configuration()` (*DataLayerInterface method*), 103

[build\\_configuration\(\) \(DllDump method\), 257](#)  
[build\\_configuration\(\) \(DllList method\), 259](#)  
[build\\_configuration\(\) \(DriverIrp method\), 260](#)  
[build\\_configuration\(\) \(DriverScan method\), 262](#)  
[build\\_configuration\(\) \(Elfs method\), 209](#)  
[build\\_configuration\(\) \(FileLayer method\), 158](#)  
[build\\_configuration\(\) \(FileScan method\), 264](#)  
[build\\_configuration\(\) \(Handles method\), 265](#)  
[build\\_configuration\(\) \(HiveList method\), 247](#)  
[build\\_configuration\(\) \(HiveScan method\), 249](#)  
[build\\_configuration\(\) \(Ifconfig method\), 227](#)  
[build\\_configuration\(\) \(Info method\), 267](#)  
[build\\_configuration\(\) \(Intel method\), 129](#)  
[build\\_configuration\(\) \(Intel32e method\), 131](#)  
[build\\_configuration\(\) \(IntelPAE method\), 134](#)  
[build\\_configuration\(\) \(IntermediateSymbol-Table method\), 402](#)  
[build\\_configuration\(\) \(ISFormatTable method\), 399](#)  
[build\\_configuration\(\) \(KernelPDBScanner method\), 45](#)  
[build\\_configuration\(\) \(LayerListRequirement method\), 67](#)  
[build\\_configuration\(\) \(LayerStacker method\), 49](#)  
[build\\_configuration\(\) \(LayerWriter method\), 304](#)  
[build\\_configuration\(\) \(LimeLayer method\), 145](#)  
[build\\_configuration\(\) \(LinearlyMappedLayer method\), 148](#)  
[build\\_configuration\(\) \(LinuxBannerCache method\), 37](#)  
[build\\_configuration\(\) \(LinuxKernelInter-medSymbols method\), 320](#)  
[build\\_configuration\(\) \(LinuxSymbolFinder method\), 39](#)  
[build\\_configuration\(\) \(Lsmode method\), 211, 229](#)  
[build\\_configuration\(\) \(Lsof method\), 212](#)  
[build\\_configuration\(\) \(lsof method\), 231](#)  
[build\\_configuration\(\) \(MacBannerCache method\), 41](#)  
[build\\_configuration\(\) \(MacKernelInter-medSymbols method\), 342](#)  
[build\\_configuration\(\) \(MacSymbolFinder method\), 42](#)  
[build\\_configuration\(\) \(Malfind method\), 214, 232, 269](#)  
[build\\_configuration\(\) \(Maps method\), 216, 235](#)  
[build\\_configuration\(\) \(ModDump method\), 271](#)  
[build\\_configuration\(\) \(ModScan method\), 273](#)  
[build\\_configuration\(\) \(Modules method\), 275](#)  
[build\\_configuration\(\) \(MutantScan method\), 277](#)  
[build\\_configuration\(\) \(Netstat method\), 234](#)  
[build\\_configuration\(\) \(PdbMSFStream method\), 150](#)  
[build\\_configuration\(\) \(PdbMultiStreamFormat method\), 153](#)  
[build\\_configuration\(\) \(PluginInterface method\), 113](#)  
[build\\_configuration\(\) \(PoolScanner method\), 279](#)  
[build\\_configuration\(\) \(PrintKey method\), 251](#)  
[build\\_configuration\(\) \(ProcDump method\), 282](#)  
[build\\_configuration\(\) \(Psaux method\), 237](#)  
[build\\_configuration\(\) \(PsList method\), 217, 238, 284](#)  
[build\\_configuration\(\) \(PsScan method\), 286](#)  
[build\\_configuration\(\) \(PsTree method\), 219, 240, 288](#)  
[build\\_configuration\(\) \(RegistryHive method\), 160](#)  
[build\\_configuration\(\) \(SegmentedLayer method\), 164](#)  
[build\\_configuration\(\) \(SSDT method\), 290](#)  
[build\\_configuration\(\) \(Statistics method\), 254](#)  
[build\\_configuration\(\) \(Strings method\), 292](#)  
[build\\_configuration\(\) \(SymbolBannerCache method\), 50](#)  
[build\\_configuration\(\) \(SymbolFinder method\), 52](#)  
[build\\_configuration\(\) \(SymbolTableInterface method\), 123](#)  
[build\\_configuration\(\) \(SymbolTableRequirement method\), 75](#)  
[build\\_configuration\(\) \(SymlinkScan method\), 294](#)  
[build\\_configuration\(\) \(Tasks method\), 241](#)  
[build\\_configuration\(\) \(Timeliner method\), 306](#)  
[build\\_configuration\(\) \(TranslationLayerInter-face method\), 107](#)  
[build\\_configuration\(\) \(TranslationLayerRe-quirement method\), 77](#)  
[build\\_configuration\(\) \(UserAssist method\), 252](#)  
[build\\_configuration\(\) \(VadDump method\), 296](#)  
[build\\_configuration\(\) \(VadInfo method\), 297](#)  
[build\\_configuration\(\) \(VerInfo method\), 299](#)  
[build\\_configuration\(\) \(Version1Format method\), 404](#)  
[build\\_configuration\(\) \(Version2Format method\), 407](#)  
[build\\_configuration\(\) \(Version3Format method\), 409](#)  
[build\\_configuration\(\) \(Version4Format method\), 412](#)  
[build\\_configuration\(\) \(Version5Format method\), 415](#)



*method*), 414  
build\_configuration() (*Version6Format method*), 417  
build\_configuration() (*Version7Format method*), 419  
build\_configuration() (*VirtMap method*), 301  
build\_configuration() (*VmwareLayer method*), 166  
build\_configuration() (*Volshell method*), 24, 26, 28, 30  
build\_configuration() (*WindowsCrash-Dump32Layer method*), 126  
build\_configuration() (*WindowsIntel method*), 136  
build\_configuration() (*WindowsIntel32e method*), 138  
build\_configuration() (*WindowsIntelPAE method*), 140  
build\_configuration() (*WindowsKernelInter-medSymbols method*), 358  
build\_configuration() (*WindowsMixin method*), 143  
build\_configuration() (*WinSwapLayers method*), 56  
build\_configuration() (*WintelHelper method*), 58  
build\_module\_collection() (*SSDT class method*), 290  
builtin\_constraints() (*PoolScanner static method*), 279  
byteorder() (*DataFormatInfo property*), 182  
Bytes (*class in volatility.framework.objects*), 175  
Bytes.VolTemplateProxy (*class in volatility.framework.objects*), 175  
BytesRequirement (*class in volatility.framework.configuration.requirements*), 61  
BytesScanner (*class in volatility.framework.layers.scanners*), 125

## C

CACHE\_PATH (*in module volatility.framework.constants*), 80  
capitalize() (*Bytes method*), 175  
capitalize() (*HexBytes method*), 312  
capitalize() (*String method*), 193  
casefold() (*String method*), 193  
cast() (*AggregateType method*), 169  
cast() (*Array method*), 170  
cast() (*BitField method*), 171  
cast() (*Boolean method*), 173  
cast() (*Bytes method*), 175  
cast() (*Char method*), 180  
cast() (*ClassType method*), 181  
cast() (*CM\_KEY\_BODY method*), 388  
cast() (*CM\_KEY\_NODE method*), 389  
cast() (*CM\_KEY\_VALUE method*), 391  
cast() (*CMHIVE method*), 387  
cast() (*dentry method*), 322  
cast() (*DEVICE\_OBJECT method*), 360  
cast() (*DRIVER\_OBJECT method*), 362  
cast() (*Enumeration method*), 183  
cast() (*EPROCESS method*), 363  
cast() (*ETHREAD method*), 365  
cast() (*EX\_FAST\_REF method*), 366  
cast() (*ExecutiveObject method*), 367  
cast() (*FILE\_OBJECT method*), 368  
cast() (*fileglob method*), 345  
cast() (*files\_struct method*), 323  
cast() (*Float method*), 185  
cast() (*fs\_struct method*), 325  
cast() (*Function method*), 187  
cast() (*GenericIntelProcess method*), 319  
cast() (*hist\_entry method*), 338  
cast() (*HMAP\_ENTRY method*), 392  
cast() (*ifnet method*), 346  
cast() (*IMAGE\_DOS\_HEADER method*), 384  
cast() (*IMAGE\_NT\_HEADERS method*), 385  
cast() (*inpcb method*), 347  
cast() (*Integer method*), 188  
cast() (*KDDEBUGGER\_DATA64 method*), 382  
cast() (*KMUTANT method*), 370  
cast() (*KSYSTEM\_TIME method*), 371  
cast() (*LIST\_ENTRY method*), 372  
cast() (*list\_head method*), 326  
cast() (*mm\_struct method*), 327  
cast() (*MMVAD method*), 374  
cast() (*MMVAD\_SHORT method*), 375  
cast() (*module method*), 328  
cast() (*mount method*), 329  
cast() (*OBJECT\_HEADER method*), 377  
cast() (*OBJECT\_SYMBOLIC\_LINK method*), 378  
cast() (*ObjectInterface method*), 110  
cast() (*Pointer method*), 190  
cast() (*POOL\_HEADER method*), 380  
cast() (*PrimitiveObject method*), 192  
cast() (*proc method*), 348  
cast() (*qstr method*), 331  
cast() (*queue\_entry method*), 349  
cast() (*SERVICE\_HEADER method*), 394  
cast() (*SERVICE\_RECORD method*), 395  
cast() (*sockaddr method*), 351  
cast() (*sockaddr\_dl method*), 352  
cast() (*socket method*), 353  
cast() (*String method*), 193  
cast() (*struct\_file method*), 332  
cast() (*StructType method*), 199  
cast() (*super\_block method*), 333



`cast()` (*task\_struct* method), 334  
`cast()` (*UNICODE\_STRING* method), 381  
`cast()` (*UnionType* method), 200  
`cast()` (*vfsmount* method), 336  
`cast()` (*vm\_area\_struct* method), 337  
`cast()` (*vm\_map\_entry* method), 354  
`cast()` (*vm\_map\_object* method), 356  
`cast()` (*vnode* method), 357  
`cast()` (*Void* method), 201  
`center()` (*Bytes* method), 175  
`center()` (*HexBytes* method), 312  
`center()` (*String* method), 194  
Certificates (class in *volatility.plugins.windows.registry.certificates*), 245  
`change_layer()` (*Volshell* method), 24, 26, 28, 31  
`change_process()` (*Volshell* method), 31  
`change_task()` (*Volshell* method), 26, 28  
CHANGED (*TimeLinerType* attribute), 306  
Char (class in *volatility.framework.objects*), 179  
Char.VolTemplateProxy (class in *volatility.framework.objects*), 179  
Check\_afinfo (class in *volatility.plugins.linux.check\_afinfo*), 206  
`check_cycles()` (*LayerContainer* method), 105  
`check_kernel_offset()` (*KernelPDBScanner* method), 45  
Check\_syscall (class in *volatility.plugins.linux.check\_syscall*), 207  
Check\_syscall (class in *volatility.plugins.mac.check\_syscall*), 223  
Check\_syscall (class in *volatility.plugins.mac.trustedbsd*), 243  
Check\_sysctl (class in *volatility.plugins.mac.check\_sysctl*), 224  
Check\_trap\_table (class in *volatility.plugins.mac.check\_trap\_table*), 226  
`children()` (*AggregateType.VolTemplateProxy* class method), 169  
`children()` (*Array.VolTemplateProxy* class method), 170  
`children()` (*BitField.VolTemplateProxy* class method), 171  
`children()` (*Boolean.VolTemplateProxy* class method), 173  
`children()` (*Bytes.VolTemplateProxy* class method), 175  
`children()` (*Char.VolTemplateProxy* class method), 179  
`children()` (*ClassType.VolTemplateProxy* class method), 181  
`children()` (*CM\_KEY\_BODY.VolTemplateProxy* class method), 388  
`children()` (*CM\_KEY\_NODE.VolTemplateProxy* class method), 389  
`children()` (*CM\_KEY\_VALUE.VolTemplateProxy* class method), 390  
`children()` (*CMHIVE.VolTemplateProxy* class method), 386  
`children()` (*dentry.VolTemplateProxy* class method), 322  
`children()` (*DEVICE\_OBJECT.VolTemplateProxy* class method), 360  
`children()` (*DRIVER\_OBJECT.VolTemplateProxy* class method), 361  
`children()` (*Enumeration.VolTemplateProxy* class method), 183  
`children()` (*EPROCESS.VolTemplateProxy* class method), 363  
`children()` (*ETHREAD.VolTemplateProxy* class method), 364  
`children()` (*EX\_FAST\_REF.VolTemplateProxy* class method), 366  
`children()` (*ExecutiveObject.VolTemplateProxy* class method), 367  
`children()` (*FILE\_OBJECT.VolTemplateProxy* class method), 368  
`children()` (*fileglob.VolTemplateProxy* class method), 344  
`children()` (*files\_struct.VolTemplateProxy* class method), 323  
`children()` (*Float.VolTemplateProxy* class method), 185  
`children()` (*fs\_struct.VolTemplateProxy* class method), 324  
`children()` (*Function.VolTemplateProxy* class method), 187  
`children()` (*GenericIntelProcess.VolTemplateProxy* class method), 318  
`children()` (*hist\_entry.VolTemplateProxy* class method), 338  
`children()` (*HMAP\_ENTRY.VolTemplateProxy* class method), 392  
`children()` (*ifnet.VolTemplateProxy* class method), 345  
`children()` (*IMAGE\_DOS\_HEADER.VolTemplateProxy* class method), 383  
`children()` (*IMAGE\_NT\_HEADERS.VolTemplateProxy* class method), 385  
`children()` (*inpcb.VolTemplateProxy* class method), 347  
`children()` (*Integer.VolTemplateProxy* class method), 188  
`children()` (*KDDEBUGGER\_DATA64.VolTemplateProxy* class method), 382  
`children()` (*KMUTANT.VolTemplateProxy* class method), 369

`children()` (`KSYSTEM_TIME.VolTemplateProxy` class method), 371

`children()` (`LIST_ENTRY.VolTemplateProxy` class method), 372

`children()` (`list_head.VolTemplateProxy` class method), 326

`children()` (`mm_struct.VolTemplateProxy` class method), 327

`children()` (`MMVAD.VolTemplateProxy` class method), 373

`children()` (`MMVAD_SHORT.VolTemplateProxy` class method), 375

`children()` (`module.VolTemplateProxy` class method), 328

`children()` (`mount.VolTemplateProxy` class method), 329

`children()` (`OBJECT_HEADER.VolTemplateProxy` class method), 377

`children()` (`OBJECT_SYMBOLIC_LINK.VolTemplateProxy` class method), 378

`children()` (`ObjectInterface.VolTemplateProxy` class method), 110

`children()` (`ObjectTemplate` property), 202

`children()` (`Pointer.VolTemplateProxy` class method), 190

`children()` (`POOL_HEADER.VolTemplateProxy` class method), 379

`children()` (`PrimitiveObject.VolTemplateProxy` class method), 192

`children()` (`proc.VolTemplateProxy` class method), 348

`children()` (`qstr.VolTemplateProxy` class method), 330

`children()` (`queue_entry.VolTemplateProxy` class method), 349

`children()` (`ReferenceTemplate` property), 203

`children()` (`SERVICE_HEADER.VolTemplateProxy` class method), 394

`children()` (`SERVICE_RECORD.VolTemplateProxy` class method), 395

`children()` (`sockaddr.VolTemplateProxy` class method), 350

`children()` (`sockaddr_dl.VolTemplateProxy` class method), 351

`children()` (`socket.VolTemplateProxy` class method), 353

`children()` (`String.VolTemplateProxy` class method), 193

`children()` (`struct_file.VolTemplateProxy` class method), 332

`children()` (`StructType.VolTemplateProxy` class method), 198

`children()` (`super_block.VolTemplateProxy` class method), 333

`children()` (`SymbolSpace.UnresolvedTemplate` property), 316

`children()` (`task_struct.VolTemplateProxy` class method), 334

`children()` (`Template` property), 112

`children()` (`TreeGrid` method), 116, 308

`children()` (`UNICODE_STRING.VolTemplateProxy` class method), 381

`children()` (`UnionType.VolTemplateProxy` class method), 199

`children()` (`vfsmount.VolTemplateProxy` class method), 335

`children()` (`vm_area_struct.VolTemplateProxy` class method), 337

`children()` (`vm_map_entry.VolTemplateProxy` class method), 354

`children()` (`vm_map_object.VolTemplateProxy` class method), 355

`children()` (`vnnode.VolTemplateProxy` class method), 356

`children()` (`Void.VolTemplateProxy` class method), 200

`ChoiceRequirement` (class in `volatility.framework.configuration.requirements`), 62

`choices()` (`Enumeration` property), 183

`choices()` (`Flags` property), 423

`choose_automagic()` (in `module` `volatility.framework.automagic`), 35

`class_subclasses()` (in `module` `volatility.framework`), 34

`classproperty` (class in `volatility`), 21

`ClassRequirement` (class in `volatility.framework.interfaces.configuration`), 89

`ClassType` (class in `volatility.framework.objects`), 181

`ClassType.VolTemplateProxy` (class in `volatility.framework.objects`), 181

`CLIRenderer` (class in `volatility.cli.text_renderer`), 33

`clone()` (`Context` method), 81

`clone()` (`ContextInterface` method), 100

`clone()` (`HierarchicalDict` method), 95

`clone()` (`ObjectTemplate` method), 202

`clone()` (`ReferenceTemplate` method), 203

`clone()` (`SymbolSpace.UnresolvedTemplate` method), 316

`clone()` (`Template` method), 112

`close()` (`JarHandler` method), 163

`cls()` (`ClassRequirement` property), 89

`CM_KEY_BODY` (class in `volatility.framework.symbols.windows.extensions.registry`), 387

`CM_KEY_BODY.VolTemplateProxy` (class in `volatility.framework.symbols.windows.extensions.registry`),

388  
 CM\_KEY\_NODE (class in volatility.framework.symbols.windows.extensions.registry), 389  
 CM\_KEY\_NODE.VolTemplateProxy (class in volatility.framework.symbols.windows.extensions.registry), 389  
 CM\_KEY\_VALUE (class in volatility.framework.symbols.windows.extensions.registry), 390  
 CM\_KEY\_VALUE.VolTemplateProxy (class in volatility.framework.symbols.windows.extensions.registry), 390  
 CmdLine (class in volatility.plugins.windows.cmdline), 256  
 CMHIVE (class in volatility.framework.symbols.windows.extensions.registry), 386  
 CMHIVE.VolTemplateProxy (class in volatility.framework.symbols.windows.extensions.registry), 386  
 Column (class in volatility.framework.interfaces.renderers), 115  
 columns() (TreeGrid property), 116, 308  
 ColumnSortKey (class in volatility.framework.interfaces.renderers), 115  
 ColumnSortKey (class in volatility.framework.renderers), 307  
 CommandLine (class in volatility.cli), 22  
 ComplexListRequirement (class in volatility.framework.configuration.requirements), 64  
 config() (AutomagicInterface property), 87  
 config() (Bash property), 205, 221  
 config() (BashIntermedSymbols property), 340  
 config() (BufferDataLayer property), 155  
 config() (Certificates property), 245  
 config() (Check\_afinfo property), 206  
 config() (Check\_syscall property), 208, 223, 243  
 config() (Check\_sysctl property), 224  
 config() (Check\_trap\_table property), 226  
 config() (CmdLine property), 256  
 config() (ConfigurableInterface property), 90  
 config() (ConfigWriter property), 303  
 config() (ConstructionMagic property), 36  
 config() (Context property), 81  
 config() (ContextInterface property), 100  
 config() (DataLayerInterface property), 103  
 config() (DllDump property), 257  
 config() (DllList property), 259  
 config() (DriverIrp property), 260  
 config() (DriverScan property), 262  
 config() (Elfs property), 209  
 config() (FileLayer property), 158  
 config() (FileScan property), 264  
 config() (Handles property), 265  
 config() (HiveList property), 247  
 config() (HiveScan property), 249  
 config() (Ifconfig property), 227  
 config() (Info property), 267  
 config() (Intel property), 129  
 config() (Intel32e property), 131  
 config() (IntelPAE property), 134  
 config() (IntermediateSymbolTable property), 402  
 config() (ISFormatTable property), 399  
 config() (KernelPDBScanner property), 45  
 config() (LayerStacker property), 49  
 config() (LayerWriter property), 304  
 config() (LimeLayer property), 145  
 config() (LinearlyMappedLayer property), 148  
 config() (LinuxBannerCache property), 38  
 config() (LinuxKernelIntermedSymbols property), 320  
 config() (LinuxSymbolFinder property), 39  
 config() (Lsmmod property), 211, 229  
 config() (Lsof property), 213  
 config() (Lsof property), 231  
 config() (MacBannerCache property), 41  
 config() (MacKernelIntermedSymbols property), 342  
 config() (MacSymbolFinder property), 43  
 config() (Malfind property), 214, 232, 269  
 config() (Maps property), 216, 235  
 config() (ModDump property), 271  
 config() (ModScan property), 273  
 config() (Modules property), 275  
 config() (MutantScan property), 277  
 config() (Netstat property), 234  
 config() (PdbMSFStream property), 150  
 config() (PdbMultiStreamFormat property), 153  
 config() (PluginInterface property), 113  
 config() (PoolScanner property), 279  
 config() (PrintKey property), 251  
 config() (ProcDump property), 282  
 config() (Psaux property), 237  
 config() (PsList property), 217, 238, 284  
 config() (PsScan property), 286  
 config() (PsTree property), 219, 240, 288  
 config() (RegistryHive property), 160  
 config() (SegmentedLayer property), 164  
 config() (SSDT property), 291  
 config() (Statistics property), 254  
 config() (Strings property), 292  
 config() (SymbolBannerCache property), 51  
 config() (SymbolFinder property), 52  
 config() (SymbolTableInterface property), 123  
 config() (SymlinkScan property), 294

`config()` (*Tasks* property), 242  
`config()` (*Timeliner* property), 306  
`config()` (*TranslationLayerInterface* property), 107  
`config()` (*UserAssist* property), 253  
`config()` (*VadDump* property), 296  
`config()` (*VadInfo* property), 297  
`config()` (*VerInfo* property), 299  
`config()` (*Version1Format* property), 404  
`config()` (*Version2Format* property), 407  
`config()` (*Version3Format* property), 409  
`config()` (*Version4Format* property), 412  
`config()` (*Version5Format* property), 414  
`config()` (*Version6Format* property), 417  
`config()` (*Version7Format* property), 419  
`config()` (*VirtMap* property), 301  
`config()` (*VmwareLayer* property), 166  
`config()` (*Volshell* property), 24, 26, 28, 31  
`config()` (*WindowsCrashDump32Layer* property), 126  
`config()` (*WindowsIntel* property), 136  
`config()` (*WindowsIntel32e* property), 138  
`config()` (*WindowsIntelPAE* property), 140  
`config()` (*WindowsKernelIntermedSymbols* property), 358  
`config()` (*WindowsMixin* property), 143  
`config()` (*WinSwapLayers* property), 56  
`config()` (*WintelHelper* property), 58  
`config_path()` (*AutomagicInterface* property), 87  
`config_path()` (*Bash* property), 205, 221  
`config_path()` (*BashIntermedSymbols* property), 340  
`config_path()` (*BufferDataLayer* property), 156  
`config_path()` (*Certificates* property), 245  
`config_path()` (*Check\_afinfo* property), 206  
`config_path()` (*Check\_syscall* property), 208, 223, 243  
`config_path()` (*Check\_sysctl* property), 224  
`config_path()` (*Check\_trap\_table* property), 226  
`config_path()` (*CmdLine* property), 256  
`config_path()` (*ConfigurableInterface* property), 90  
`config_path()` (*ConfigWriter* property), 303  
`config_path()` (*ConstructionMagic* property), 36  
`config_path()` (*DataLayerInterface* property), 103  
`config_path()` (*DllDump* property), 257  
`config_path()` (*DllList* property), 259  
`config_path()` (*DriverIrp* property), 261  
`config_path()` (*DriverScan* property), 262  
`config_path()` (*Elfs* property), 209  
`config_path()` (*FileLayer* property), 158  
`config_path()` (*FileScan* property), 264  
`config_path()` (*Handles* property), 265  
`config_path()` (*HiveList* property), 247  
`config_path()` (*HiveScan* property), 249  
`config_path()` (*Ifconfig* property), 228  
`config_path()` (*Info* property), 267  
`config_path()` (*Intel* property), 129  
`config_path()` (*Intel32e* property), 131  
`config_path()` (*IntelPAE* property), 134  
`config_path()` (*IntermediateSymbolTable* property), 402  
`config_path()` (*ISFormatTable* property), 399  
`config_path()` (*KernelPDBScanner* property), 45  
`config_path()` (*LayerStacker* property), 49  
`config_path()` (*LayerWriter* property), 304  
`config_path()` (*LimeLayer* property), 145  
`config_path()` (*LinearlyMappedLayer* property), 148  
`config_path()` (*LinuxBannerCache* property), 38  
`config_path()` (*LinuxKernelIntermedSymbols* property), 320  
`config_path()` (*LinuxSymbolFinder* property), 39  
`config_path()` (*Lsmmod* property), 211, 229  
`config_path()` (*Lsof* property), 213  
`config_path()` (*Isof* property), 231  
`config_path()` (*MacBannerCache* property), 41  
`config_path()` (*MacKernelIntermedSymbols* property), 342  
`config_path()` (*MacSymbolFinder* property), 43  
`config_path()` (*Malfind* property), 214, 232, 269  
`config_path()` (*Maps* property), 216, 235  
`config_path()` (*ModDump* property), 271  
`config_path()` (*ModScan* property), 273  
`config_path()` (*Modules* property), 275  
`config_path()` (*MutantScan* property), 277  
`config_path()` (*Netstat* property), 234  
`config_path()` (*PdbMSFStream* property), 150  
`config_path()` (*PdbMultiStreamFormat* property), 153  
`config_path()` (*PluginInterface* property), 113  
`config_path()` (*PoolScanner* property), 279  
`config_path()` (*PrintKey* property), 251  
`config_path()` (*ProcDump* property), 283  
`config_path()` (*Psaux* property), 237  
`config_path()` (*PsList* property), 217, 238, 284  
`config_path()` (*PsScan* property), 286  
`config_path()` (*PsTree* property), 219, 240, 288  
`config_path()` (*RegistryHive* property), 160  
`config_path()` (*SegmentedLayer* property), 164  
`config_path()` (*SSDT* property), 291  
`config_path()` (*Statistics* property), 254  
`config_path()` (*Strings* property), 292  
`config_path()` (*SymbolBannerCache* property), 51  
`config_path()` (*SymbolFinder* property), 52  
`config_path()` (*SymbolTableInterface* property), 123  
`config_path()` (*SymlinkScan* property), 294  
`config_path()` (*Tasks* property), 242  
`config_path()` (*Timeliner* property), 306



- `config_path()` (*TranslationLayerInterface* property), 107
- `config_path()` (*UserAssist* property), 253
- `config_path()` (*VadDump* property), 296
- `config_path()` (*VadInfo* property), 297
- `config_path()` (*VerInfo* property), 299
- `config_path()` (*Version1Format* property), 404
- `config_path()` (*Version2Format* property), 407
- `config_path()` (*Version3Format* property), 409
- `config_path()` (*Version4Format* property), 412
- `config_path()` (*Version5Format* property), 414
- `config_path()` (*Version6Format* property), 417
- `config_path()` (*Version7Format* property), 420
- `config_path()` (*VirtMap* property), 301
- `config_path()` (*VmwareLayer* property), 166
- `config_path()` (*Volshell* property), 24, 26, 28, 31
- `config_path()` (*WindowsCrashDump32Layer* property), 126
- `config_path()` (*WindowsIntel* property), 136
- `config_path()` (*WindowsIntel32e* property), 138
- `config_path()` (*WindowsIntelPAE* property), 141
- `config_path()` (*WindowsKernelIntermedSymbols* property), 358
- `config_path()` (*WindowsMixin* property), 143
- `config_path()` (*WinSwapLayers* property), 56
- `config_path()` (*WintelHelper* property), 58
- `CONFIG_SEPARATOR` (in module *volatility.framework.interfaces.configuration*), 89
- `config_value()` (*BooleanRequirement* method), 60
- `config_value()` (*BytesRequirement* method), 61
- `config_value()` (*ChoiceRequirement* method), 63
- `config_value()` (*ClassRequirement* method), 89
- `config_value()` (*ComplexListRequirement* method), 64
- `config_value()` (*ConfigurableRequirementInterface* method), 92
- `config_value()` (*ConstructableRequirementInterface* method), 93
- `config_value()` (*IntRequirement* method), 66
- `config_value()` (*LayerListRequirement* method), 67
- `config_value()` (*ListRequirement* method), 69
- `config_value()` (*MultiRequirement* method), 71
- `config_value()` (*PluginRequirement* method), 72
- `config_value()` (*RequirementInterface* method), 96
- `config_value()` (*SimpleTypeRequirement* method), 98
- `config_value()` (*StringRequirement* method), 74
- `config_value()` (*SymbolTableRequirement* method), 75
- `config_value()` (*TranslationLayerRequirement* method), 77
- `config_value()` (*URIRequirement* method), 78
- ConfigurableInterface* (class in *volatility.framework.interfaces.configuration*), 90
- ConfigurableRequirementInterface* (class in *volatility.framework.interfaces.configuration*), 91
- ConfigWriter* (class in *volatility.plugins.configwriter*), 302
- `conjugate()` (*Bin* method), 311
- `conjugate()` (*BitField* method), 172
- `conjugate()` (*Boolean* method), 174
- `conjugate()` (*Char* method), 180
- `conjugate()` (*Enumeration* method), 183
- `conjugate()` (*Float* method), 185
- `conjugate()` (*Hex* method), 312
- `conjugate()` (*Integer* method), 188
- `conjugate()` (*Pointer* method), 190
- `constant_data()` (*SymbolInterface* property), 121
- `construct()` (*ComplexListRequirement* method), 64
- `construct()` (*ConstructableRequirementInterface* method), 93
- `construct()` (*LayerListRequirement* method), 67
- `construct()` (*SymbolTableRequirement* method), 75
- `construct()` (*TranslationLayerRequirement* method), 77
- `construct_locals()` (*Volshell* method), 24, 26, 28, 31
- ConstructableRequirementInterface* (class in *volatility.framework.interfaces.configuration*), 93
- ConstructionMagic* (class in *volatility.framework.automagic.construct\_layers*), 36
- `consume_file()` (*CommandLine* method), 22
- `consume_file()` (*FileConsumerInterface* method), 113
- `consume_file()` (*NullFileConsumer* method), 24
- `consume_file()` (*VolShell* method), 23
- `consume_file()` (*Volshell* method), 24, 26, 29, 31
- `consume_padding()` (*PdbReader* method), 397
- `consume_type()` (*PdbReader* method), 397
- Context* (class in *volatility.framework.contexts*), 81
- `context()` (*AutomagicInterface* property), 87
- `context()` (*Bash* property), 205, 221
- `context()` (*BashIntermedSymbols* property), 340
- `context()` (*BufferDataLayer* property), 156
- `context()` (*BytesScanner* property), 125
- `context()` (*Certificates* property), 246
- `context()` (*Check\_afinfo* property), 206
- `context()` (*Check\_syscall* property), 208, 223, 244
- `context()` (*Check\_sysctl* property), 225
- `context()` (*Check\_trap\_table* property), 226
- `context()` (*CmdLine* property), 256
- `context()` (*ConfigurableInterface* property), 91
- `context()` (*ConfigWriter* property), 303
- `context()` (*ConstructionMagic* property), 36
- `context()` (*DataLayerInterface* property), 103

`context ()` (*DllDump property*), 258  
`context ()` (*DllList property*), 259  
`context ()` (*DriverIrp property*), 261  
`context ()` (*DriverScan property*), 262  
`context ()` (*Elfs property*), 209  
`context ()` (*FileLayer property*), 158  
`context ()` (*FileScan property*), 264  
`context ()` (*Handles property*), 265  
`context ()` (*HiveList property*), 247  
`context ()` (*HiveScan property*), 249  
`context ()` (*Ifconfig property*), 228  
`context ()` (*Info property*), 268  
`context ()` (*Intel property*), 129  
`context ()` (*Intel32e property*), 132  
`context ()` (*IntelPAE property*), 134  
`context ()` (*IntermediateSymbolTable property*), 402  
`context ()` (*ISFormatTable property*), 399  
`context ()` (*KernelPDBScanner property*), 45  
`context ()` (*LayerStacker property*), 49  
`context ()` (*LayerWriter property*), 304  
`context ()` (*LimeLayer property*), 145  
`context ()` (*LinearlyMappedLayer property*), 148  
`context ()` (*LinuxBannerCache property*), 38  
`context ()` (*LinuxKernelIntermedSymbols property*), 320  
`context ()` (*LinuxSymbolFinder property*), 39  
`context ()` (*Lsmmod property*), 211, 229  
`context ()` (*Lsof property*), 213  
`context ()` (*lsof property*), 231  
`context ()` (*MacBannerCache property*), 41  
`context ()` (*MacKernelIntermedSymbols property*), 342  
`context ()` (*MacSymbolFinder property*), 43  
`context ()` (*Malfind property*), 214, 232, 269  
`context ()` (*Maps property*), 216, 235  
`context ()` (*ModDump property*), 271  
`context ()` (*ModScan property*), 273  
`context ()` (*Module property*), 83  
`context ()` (*ModuleInterface property*), 101  
`context ()` (*Modules property*), 275  
`context ()` (*MultiStringScanner property*), 125  
`context ()` (*MutantScan property*), 277  
`context ()` (*Netstat property*), 234  
`context ()` (*PageMapScanner property*), 56  
`context ()` (*PdbMSFStream property*), 151  
`context ()` (*PdbMultiStreamFormat property*), 153  
`context ()` (*PdbReader property*), 397  
`context ()` (*PdbSignatureScanner property*), 48  
`context ()` (*PluginInterface property*), 113  
`context ()` (*PoolHeaderScanner property*), 278  
`context ()` (*PoolScanner property*), 279  
`context ()` (*PrintKey property*), 251  
`context ()` (*ProcDump property*), 283  
`context ()` (*Psaux property*), 237  
`context ()` (*PsList property*), 217, 238, 284  
`context ()` (*PsScan property*), 287  
`context ()` (*PsTree property*), 219, 240, 288  
`context ()` (*RegExScanner property*), 125  
`context ()` (*RegistryHive property*), 160  
`context ()` (*ScannerInterface property*), 107  
`context ()` (*SegmentedLayer property*), 164  
`context ()` (*SizedModule property*), 84  
`context ()` (*SSDT property*), 291  
`context ()` (*Statistics property*), 254  
`context ()` (*Strings property*), 292  
`context ()` (*SymbolBannerCache property*), 51  
`context ()` (*SymbolFinder property*), 52  
`context ()` (*SymbolTableInterface property*), 123  
`context ()` (*SymlinkScan property*), 294  
`context ()` (*Tasks property*), 242  
`context ()` (*Timeliner property*), 306  
`context ()` (*TranslationLayerInterface property*), 107  
`context ()` (*UserAssist property*), 253  
`context ()` (*VadDump property*), 296  
`context ()` (*VadInfo property*), 297  
`context ()` (*VerInfo property*), 299  
`context ()` (*Version1Format property*), 405  
`context ()` (*Version2Format property*), 407  
`context ()` (*Version3Format property*), 410  
`context ()` (*Version4Format property*), 412  
`context ()` (*Version5Format property*), 415  
`context ()` (*Version6Format property*), 417  
`context ()` (*Version7Format property*), 420  
`context ()` (*VirtMap property*), 301  
`context ()` (*VmwareLayer property*), 166  
`context ()` (*Volshell property*), 24, 26, 29, 31  
`context ()` (*WindowsCrashDump32Layer property*), 126  
`context ()` (*WindowsIntel property*), 136  
`context ()` (*WindowsIntel32e property*), 138  
`context ()` (*WindowsIntelPAE property*), 141  
`context ()` (*WindowsKernelIntermedSymbols property*), 358  
`context ()` (*WindowsMixin property*), 143  
`context ()` (*WinSwapLayers property*), 57  
`context ()` (*WintelHelper property*), 58  
`ContextInterface` (class in `volatility.framework.interfaces.context`), 99  
`convert_bytes_to_guid ()` (*PdbReader method*), 397  
`convert_data_to_value ()` (in module `volatility.framework.objects`), 201  
`convert_fields ()` (*PdbReader method*), 397  
`convert_ip_v4 ()` (in module `volatility.framework.renderers.conversion`), 310  
`convert_ip_v6 ()` (in module `volatility.framework.renderers.conversion`), 310  
`convert_network_four_tuple ()` (in module

volatility.framework.renderers.conversion),  
 310  
 convert\_port() (in module volatility.framework.renderers.conversion), 310  
 convert\_value\_to\_data() (in module volatility.framework.objects), 201  
 count() (Array property), 170  
 count() (Bytes method), 175  
 count() (Column method), 115  
 count() (DataFormatInfo method), 182  
 count() (HexBytes method), 312  
 count() (String method), 194  
 count() (TreeNode method), 117, 309  
 create() (BashIntermedSymbols class method), 340  
 create() (IntermediateSymbolTable class method), 402  
 create() (LinuxKernelIntermedSymbols class method), 320  
 create() (MacKernelIntermedSymbols class method), 342  
 create() (WindowsKernelIntermedSymbols class method), 358  
 create\_json\_hash() (in module volatility.schemas), 425  
 create\_name\_filter() (PsList class method), 284  
 create\_name\_filter() (PsTree class method), 288  
 create\_pid\_filter() (PsList class method), 217, 238, 284  
 create\_pid\_filter() (PsTree class method), 219, 288  
 create\_pid\_filter() (Tasks class method), 242  
 create\_stream\_from\_pages() (PdbMultiStreamFormat method), 153  
 CREATED (TimeLinerType attribute), 306  
 CSVRenderer (class in volatility.cli.text\_renderer), 33  
 current\_layer() (Volshell property), 24, 26, 29, 31

## D

data() (HierarchicalDict property), 95  
 DataFormatInfo (class in volatility.framework.objects), 182  
 DataLayerInterface (class in volatility.framework.interfaces.layers), 103  
 decode() (Bytes method), 175  
 decode() (HexBytes method), 313  
 decode\_data() (CM\_KEY\_VALUE method), 391  
 deduplicate() (ModuleCollection method), 84  
 default() (BooleanRequirement property), 60  
 default() (BytesRequirement property), 61  
 default() (ChoiceRequirement property), 63  
 default() (ClassRequirement property), 89  
 default() (ComplexListRequirement property), 64  
 default() (ConfigurableRequirementInterface property), 92  
 default() (ConstructableRequirementInterface property), 94  
 default() (IntRequirement property), 66  
 default() (LayerListRequirement property), 68  
 default() (ListRequirement property), 69  
 default() (MultiRequirement property), 71  
 default() (PluginRequirement property), 72  
 default() (RequirementInterface property), 97  
 default() (SimpleTypeRequirement property), 98  
 default() (StringRequirement property), 74  
 default() (SymbolTableRequirement property), 75  
 default() (TranslationLayerRequirement property), 77  
 default() (URIRequirement property), 79  
 default\_block\_size (LayerWriter attribute), 304  
 default\_open() (JarHandler static method), 163  
 default\_output\_name (LayerWriter attribute), 304  
 del\_layer() (LayerContainer method), 105  
 del\_type\_class() (BaseSymbolTableInterface method), 118  
 del\_type\_class() (BashIntermedSymbols method), 340  
 del\_type\_class() (IntermediateSymbolTable method), 403  
 del\_type\_class() (ISFormatTable method), 399  
 del\_type\_class() (LinuxKernelIntermedSymbols method), 320  
 del\_type\_class() (MacKernelIntermedSymbols method), 343  
 del\_type\_class() (NativeTable method), 422  
 del\_type\_class() (NativeTableInterface method), 119  
 del\_type\_class() (SymbolTableInterface method), 123  
 del\_type\_class() (Version1Format method), 405  
 del\_type\_class() (Version2Format method), 407  
 del\_type\_class() (Version3Format method), 410  
 del\_type\_class() (Version4Format method), 412  
 del\_type\_class() (Version5Format method), 415  
 del\_type\_class() (Version6Format method), 417  
 del\_type\_class() (Version7Format method), 420  
 del\_type\_class() (WindowsKernelIntermedSymbols method), 358  
 denominator (Bin attribute), 311  
 denominator (BitField attribute), 172  
 denominator (Boolean attribute), 174  
 denominator (Char attribute), 180  
 denominator (Enumeration attribute), 183  
 denominator (Hex attribute), 312  
 denominator (Integer attribute), 188  
 denominator (Pointer attribute), 190  
 dentry (class in volatility.framework.symbols.linux.extensions), 322

`dentry.VolTemplateProxy` (class in `volatility.framework.symbols.linux.extensions`), 322

`dependencies()` (`BufferDataLayer` property), 156

`dependencies()` (`DataLayerInterface` property), 103

`dependencies()` (`FileLayer` property), 158

`dependencies()` (`Intel` property), 129

`dependencies()` (`Intel32e` property), 132

`dependencies()` (`IntelPAE` property), 134

`dependencies()` (`LimeLayer` property), 146

`dependencies()` (`LinearlyMappedLayer` property), 148

`dependencies()` (`PdbMSFStream` property), 151

`dependencies()` (`PdbMultiStreamFormat` property), 153

`dependencies()` (`RegistryHive` property), 160

`dependencies()` (`SegmentedLayer` property), 164

`dependencies()` (`TranslationLayerInterface` property), 107

`dependencies()` (`VmwareLayer` property), 166

`dependencies()` (`WindowsCrashDump32Layer` property), 127

`dependencies()` (`WindowsIntel` property), 136

`dependencies()` (`WindowsIntel32e` property), 138

`dependencies()` (`WindowsIntelPAE` property), 141

`dependencies()` (`WindowsMixin` property), 143

`dereference()` (`EX_FAST_REF` method), 366

`dereference()` (`Pointer` method), 190

`description()` (`BooleanRequirement` property), 60

`description()` (`BytesRequirement` property), 62

`description()` (`ChoiceRequirement` property), 63

`description()` (`ClassRequirement` property), 89

`description()` (`ComplexListRequirement` property), 65

`description()` (`ConfigurableRequirementInterface` property), 92

`description()` (`ConstructableRequirementInterface` property), 94

`description()` (`Enumeration` property), 183

`description()` (`IntRequirement` property), 66

`description()` (`LayerListRequirement` property), 68

`description()` (`ListRequirement` property), 69

`description()` (`MultiRequirement` property), 71

`description()` (`PluginRequirement` property), 72

`description()` (`RequirementInterface` property), 97

`description()` (`SimpleTypeRequirement` property), 98

`description()` (`StringRequirement` property), 74

`description()` (`SymbolTableRequirement` property), 75

`description()` (`TranslationLayerRequirement` property), 77

`description()` (`URIRequirement` property), 79

`destroy()` (`BufferDataLayer` method), 156

`destroy()` (`DataLayerInterface` method), 103

`destroy()` (`FileLayer` method), 158

`destroy()` (`Intel` method), 129

`destroy()` (`Intel32e` method), 132

`destroy()` (`IntelPAE` method), 134

`destroy()` (`LimeLayer` method), 146

`destroy()` (`LinearlyMappedLayer` method), 148

`destroy()` (`PdbMSFStream` method), 151

`destroy()` (`PdbMultiStreamFormat` method), 153

`destroy()` (`RegistryHive` method), 160

`destroy()` (`SegmentedLayer` method), 164

`destroy()` (`TranslationLayerInterface` method), 107

`destroy()` (`VmwareLayer` method), 166

`destroy()` (`WindowsCrashDump32Layer` method), 127

`destroy()` (`WindowsIntel` method), 136

`destroy()` (`WindowsIntel32e` method), 138

`destroy()` (`WindowsIntelPAE` method), 141

`destroy()` (`WindowsMixin` method), 143

`determine_extended_value()` (`PdbReader` method), 397

`determine_map()` (`VirtMap` class method), 301

`determine_valid_kernels()` (`KernelPDBScanner` method), 45

`DEVICE_OBJECT` (class in `volatility.framework.symbols.windows.extensions`), 360

`DEVICE_OBJECT.VolTemplateProxy` (class in `volatility.framework.symbols.windows.extensions`), 360

`disassemble()` (`Volshell` method), 24, 27, 29, 31

`Disassembly` (class in `volatility.framework.interfaces.renderers`), 115

`display_bytes()` (`Volshell` method), 24, 27, 29, 31

`display_disassembly()` (in module `volatility.cli.text_renderer`), 34

`display_doublewords()` (`Volshell` method), 24, 27, 29, 31

`display_plugin_output()` (`Volshell` method), 24, 27, 29, 31

`display_quadwords()` (`Volshell` method), 25, 27, 29, 31

`display_symbols()` (`Volshell` method), 25, 27, 29, 31

`display_type()` (`Volshell` method), 25, 27, 29, 31

`display_words()` (`Volshell` method), 25, 27, 29, 31

`DllDump` (class in `volatility.plugins.windows.dlldump`), 257

`DllList` (class in `volatility.plugins.windows.dlllist`), 259

`download_pdb_isf()` (`KernelPDBScanner` method), 45

`DRIVER_OBJECT` (class in `volatility.framework.symbols.windows.extensions`), 360



361  
 DRIVER\_OBJECT.VolTemplateProxy  
     (class in volatility.framework.symbols.windows.extensions),  
     361  
 DriverIrp (class in volatility.plugins.windows.driverirp), 260  
 DriverScan (class in volatility.plugins.windows.driverscan), 262  
 DtbSelfRef32bit (class in volatility.framework.automagic.windows), 54  
 DtbSelfRef64bit (class in volatility.framework.automagic.windows), 54  
 DtbSelfReferential (class in volatility.framework.automagic.windows), 54  
 DtbTest (class in volatility.framework.automagic.windows), 55  
 DtbTest32bit (class in volatility.framework.automagic.windows), 55  
 DtbTest64bit (class in volatility.framework.automagic.windows), 55  
 DtbTestPae (class in volatility.framework.automagic.windows), 56  
 DummyLock (class in volatility.framework.layers.physical), 157  
 DummyProgress (class in volatility.framework.interfaces.layers), 105

## E

Elfs (class in volatility.plugins.linux.elfs), 209  
 encode() (String method), 194  
 endswith() (Bytes method), 176  
 endswith() (HexBytes method), 313  
 endswith() (String method), 194  
 ENUM (SymbolType attribute), 318  
 Enumeration (class in volatility.framework.objects), 182  
 Enumeration.VolTemplateProxy (class in volatility.framework.objects), 182  
 enumerations() (BaseSymbolTableInterface property), 118  
 enumerations() (BashIntermedSymbols property), 340  
 enumerations() (IntermediateSymbolTable property), 403  
 enumerations() (ISFormatTable property), 399  
 enumerations() (LinuxKernelIntermedSymbols property), 320  
 enumerations() (MacKernelIntermedSymbols property), 343  
 enumerations() (NativeTable property), 422  
 enumerations() (NativeTableInterface property), 120

enumerations() (SymbolTableInterface property), 123  
 enumerations() (Version1Format property), 405  
 enumerations() (Version2Format property), 407  
 enumerations() (Version3Format property), 410  
 enumerations() (Version4Format property), 412  
 enumerations() (Version5Format property), 415  
 enumerations() (Version6Format property), 417  
 enumerations() (Version7Format property), 420  
 enumerations() (WindowsKernelIntermedSymbols property), 359  
 environment variable  
     PYTHONPATH, 19  
 EPROCESS (class in volatility.framework.symbols.windows.extensions), 362  
 EPROCESS.VolTemplateProxy (class in volatility.framework.symbols.windows.extensions), 363  
 ETHREAD (class in volatility.framework.symbols.windows.extensions), 364  
 ETHREAD.VolTemplateProxy (class in volatility.framework.symbols.windows.extensions), 364  
 EX\_FAST\_REF (class in volatility.framework.symbols.windows.extensions), 365  
 EX\_FAST\_REF.VolTemplateProxy  
     (class in volatility.framework.symbols.windows.extensions),  
     365  
 ExecutiveObject (class in volatility.framework.symbols.windows.extensions), 366  
 ExecutiveObject.VolTemplateProxy  
     (class in volatility.framework.symbols.windows.extensions),  
     367  
 expandtabs() (Bytes method), 176  
 expandtabs() (HexBytes method), 313  
 expandtabs() (String method), 194  
 extended\_flags (vm\_area\_struct attribute), 337

## F

file\_name\_with\_device() (FILE\_OBJECT method), 369  
 FILE\_OBJECT (class in volatility.framework.symbols.windows.extensions), 368  
 FILE\_OBJECT.VolTemplateProxy  
     (class in volatility.framework.symbols.windows.extensions),  
     368

`file_symbol_url()` (*BashIntermedSymbols* class method), 340

`file_symbol_url()` (*IntermediateSymbolTable* class method), 403

`file_symbol_url()` (*LinuxKernelIntermedSymbols* class method), 320

`file_symbol_url()` (*MacKernelIntermedSymbols* class method), 343

`file_symbol_url()` (*WindowsKernelIntermedSymbols* class method), 359

`FileConsumerInterface` (class in *volatility.framework.interfaces.plugins*), 112

`fileglob` (class in *volatility.framework.symbols.mac.extensions*), 344

`fileglob.VolTemplateProxy` (class in *volatility.framework.symbols.mac.extensions*), 344

`FileInterface` (class in *volatility.framework.interfaces.plugins*), 113

`FileLayer` (class in *volatility.framework.layers.physical*), 157

`files_descriptors_for_process()` (*LinuxUtilities* class method), 40

`files_descriptors_for_process()` (*MacUtilities* class method), 44

`files_struct` (class in *volatility.framework.symbols.linux.extensions*), 323

`files_struct.VolTemplateProxy` (class in *volatility.framework.symbols.linux.extensions*), 323

`FileScan` (class in *volatility.plugins.windows.filescan*), 263

`find()` (*Bytes* method), 176

`find()` (*HexBytes* method), 313

`find()` (*String* method), 194

`find_aslr()` (*LinuxUtilities* class method), 40

`find_aslr()` (*MacUtilities* class method), 44

`find_cookie()` (*Handles* class method), 266

`find_level()` (*PsTree* method), 219, 289

`find_module()` (*WarningFindSpec* method), 21

`find_requirements()` (*AutomagicInterface* method), 87

`find_requirements()` (*ConstructionMagic* method), 36

`find_requirements()` (*KernelPDBScanner* method), 46

`find_requirements()` (*LayerStacker* method), 49

`find_requirements()` (*LinuxBannerCache* method), 38

`find_requirements()` (*LinuxSymbolFinder* method), 39

`find_requirements()` (*MacBannerCache* method), 41

`find_requirements()` (*MacSymbolFinder* method), 43

`find_requirements()` (*SymbolBannerCache* method), 51

`find_requirements()` (*SymbolFinder* method), 53

`find_requirements()` (*WinSwapLayers* method), 57

`find_requirements()` (*WintelHelper* method), 58

`find_sar_value()` (*Handles* method), 266

`find_session_layer()` (*ModDump* class method), 271

`find_spec()` (*WarningFindSpec* static method), 21

`find_suitable_requirements()` (*LayerStacker* method), 49

`find_swap_requirement()` (*WinSwapLayers* static method), 57

`find_virtual_layers_from_req()` (*KernelPDBScanner* method), 46

`fix_image_base()` (*IMAGE\_DOS\_HEADER* method), 384

`Flags` (class in *volatility.framework.symbols.wrappers*), 423

`Float` (class in *volatility.framework.objects*), 184

`Float.VolTemplateProxy` (class in *volatility.framework.objects*), 185

`format()` (*String* method), 194

`format_map()` (*String* method), 194

`format_mapping` (*Version4Format* attribute), 412

`format_mapping` (*Version5Format* attribute), 415

`format_mapping` (*Version6Format* attribute), 417

`format_mapping` (*Version7Format* attribute), 420

`ForwardArrayCount` (class in *volatility.framework.symbols.windows.pdbconv*), 397

`FREE` (*PoolType* attribute), 281

`free_layer_name()` (*LayerContainer* method), 106

`free_table_name()` (*SymbolSpace* method), 317

`free_table_name()` (*SymbolSpaceInterface* method), 121

`from_bytes()` (*Bin* method), 311

`from_bytes()` (*BitField* method), 172

`from_bytes()` (*Boolean* method), 174

`from_bytes()` (*Char* method), 180

`from_bytes()` (*Enumeration* method), 183

`from_bytes()` (*Hex* method), 312

`from_bytes()` (*Integer* method), 189

`from_bytes()` (*Pointer* method), 191

`fromhex()` (*Bytes* method), 176

`fromhex()` (*Float* method), 185

`fromhex()` (*HexBytes* method), 313

`fs_struct` (class in *volatility.framework.symbols.linux.extensions*), 324

`fs_struct.VolTemplateProxy` (class in *volatility.framework.symbols.linux.extensions*), 324

`full_path()` (*vnnode method*), 357  
 Function (*class in volatility.framework.objects*), 186  
 Function.VolTemplateProxy (*class in volatility.framework.objects*), 186

## G

`generate_mapping()` (*Strings method*), 292  
`generate_pool_scan()` (*PoolScanner class method*), 279  
`generate_timeline()` (*Bash method*), 205, 222  
`generate_timeline()` (*PsList method*), 285  
`generate_timeline()` (*PsScan method*), 287  
`generate_timeline()` (*PsTree method*), 289  
`generate_timeline()` (*SymlinkScan method*), 294  
`generate_timeline()` (*TimeLinerInterface method*), 305  
`generate_treegrid()` (*Volshell method*), 25, 27, 29, 31  
`generator()` (*HierarchicalDict method*), 95  
 GenericIntelProcess (*class in volatility.framework.symbols.generic*), 318  
 GenericIntelProcess.VolTemplateProxy (*class in volatility.framework.symbols.generic*), 318  
`get` (*RegValueTypes attribute*), 393  
`get()` (*HierarchicalDict method*), 95  
`get()` (*LayerContainer method*), 106  
`get()` (*ObjectInformation method*), 110  
`get()` (*ReadOnlyMapping method*), 111  
`get()` (*SymbolSpace method*), 317  
`get()` (*SymbolSpaceInterface method*), 121  
`get_address()` (*sockaddr method*), 351  
`get_binary()` (*SERVICE\_RECORD method*), 396  
`get_block_offset()` (*HMAP\_ENTRY method*), 392  
`get_build_lab()` (*KDDEBUGGER\_DATA64 method*), 382  
`get_cell()` (*RegistryHive method*), 161  
`get_command()` (*hist\_entry method*), 339  
`get_commit_charge()` (*MMVAD method*), 374  
`get_commit_charge()` (*MMVAD\_SHORT method*), 375  
`get_connection_info()` (*socket method*), 353  
`get_converted_connection_info()` (*socket method*), 353  
`get_core_size()` (*module method*), 328  
`get_create_time()` (*EPROCESS method*), 363  
`get_create_time()` (*OBJECT\_SYMBOLIC\_LINK method*), 378  
`get_csdversion()` (*KDDEBUGGER\_DATA64 method*), 383  
`get_dentry()` (*struct\_file method*), 332  
`get_depends()` (*Info class method*), 268

`get_device_name()` (*DEVICE\_OBJECT method*), 361  
`get_display()` (*SERVICE\_RECORD method*), 396  
`get_driver_name()` (*DRIVER\_OBJECT method*), 362  
`get_end()` (*MMVAD method*), 374  
`get_end()` (*MMVAD\_SHORT method*), 375  
`get_enumeration()` (*BashIntermedSymbols method*), 340  
`get_enumeration()` (*IntermediateSymbolTable method*), 403  
`get_enumeration()` (*LinuxKernelIntermedSymbols method*), 321  
`get_enumeration()` (*MacKernelIntermedSymbols method*), 343  
`get_enumeration()` (*Module method*), 83  
`get_enumeration()` (*ModuleInterface method*), 101  
`get_enumeration()` (*NativeTable method*), 422  
`get_enumeration()` (*NativeTableInterface method*), 120  
`get_enumeration()` (*SizedModule method*), 85  
`get_enumeration()` (*SymbolSpace method*), 317  
`get_enumeration()` (*SymbolSpaceInterface method*), 121  
`get_enumeration()` (*Version1Format method*), 405  
`get_enumeration()` (*Version2Format method*), 407  
`get_enumeration()` (*Version3Format method*), 410  
`get_enumeration()` (*Version4Format method*), 412  
`get_enumeration()` (*Version5Format method*), 415  
`get_enumeration()` (*Version6Format method*), 417  
`get_enumeration()` (*Version7Format method*), 420  
`get_enumeration()` (*WindowsKernelIntermedSymbols method*), 359  
`get_exit_time()` (*EPROCESS method*), 363  
`get_family()` (*socket method*), 353  
`get_fds()` (*files\_struct method*), 324  
`get_fg_type()` (*fileglob method*), 345  
`get_file_name()` (*MMVAD method*), 374  
`get_file_name()` (*MMVAD\_SHORT method*), 375  
`get_flags()` (*vm\_area\_struct method*), 337  
`get_full_key_name()` (*CM\_KEY\_BODY method*), 388  
`get_handle_count()` (*EPROCESS method*), 363  
`get_init_size()` (*module method*), 328  
`get_inpcb()` (*socket method*), 353  
`get_ipv4_info()` (*inpcb method*), 347  
`get_ipv6_info()` (*inpcb method*), 347  
`get_is_wow64()` (*EPROCESS method*), 363  
`get_json()` (*PdbReader method*), 397  
`get_key()` (*RegistryHive method*), 161  
`get_key_path()` (*CM\_KEY\_NODE method*), 389  
`get_left_child()` (*MMVAD method*), 374  
`get_left_child()` (*MMVAD\_SHORT method*), 375

`get_link_name()` (*OBJECT\_SYMBOLIC\_LINK method*), 378

`get_map_iter()` (*proc method*), 348

`get_map_object()` (*vm\_map\_object method*), 356

`get_max_fds()` (*files\_struct method*), 324

`get_mmap_iter()` (*mm\_struct method*), 327

`get_mnt_flags()` (*mount method*), 330

`get_mnt_mountpoint()` (*mount method*), 330

`get_mnt_mountpoint()` (*vfsmount method*), 336

`get_mnt_parent()` (*mount method*), 330

`get_mnt_parent()` (*vfsmount method*), 336

`get_mnt_root()` (*mount method*), 330

`get_mnt_root()` (*vfsmount method*), 336

`get_mnt_sb()` (*mount method*), 330

`get_module_symbols_by_absolute_location()` (*ModuleCollection method*), 84

`get_module_wrapper()` (*in module volatility.framework.contexts*), 86

`get_name()` (*CM\_KEY\_NODE method*), 389

`get_name()` (*CM\_KEY\_VALUE method*), 391

`get_name()` (*CMHIVE method*), 387

`get_name()` (*KMUTANT method*), 370

`get_name()` (*RegistryHive method*), 161

`get_name()` (*SERVICE\_RECORD method*), 396

`get_name()` (*vm\_area\_struct method*), 337

`get_node()` (*RegistryHive method*), 161

`get_nt_header()` (*IMAGE\_DOS\_HEADER method*), 384

`get_object()` (*POOL\_HEADER method*), 380

`get_object()` (*vm\_map\_entry method*), 354

`get_object_type()` (*OBJECT\_HEADER method*), 377

`get_offset()` (*vm\_map\_entry method*), 354

`get_page_offset()` (*vm\_area\_struct method*), 337

`get_parent()` (*MMVAD method*), 374

`get_parent()` (*MMVAD\_SHORT method*), 376

`get_path()` (*vm\_map\_entry method*), 354

`get_perms()` (*vm\_map\_entry method*), 354

`get_physical_layer_name()` (*KernelPDBScanner method*), 46

`get_pid()` (*SERVICE\_RECORD method*), 396

`get_private_memory()` (*MMVAD method*), 374

`get_private_memory()` (*MMVAD\_SHORT method*), 376

`get_process_memory_sections()` (*proc method*), 348

`get_process_memory_sections()` (*task\_struct method*), 335

`get_protection()` (*MMVAD method*), 374

`get_protection()` (*MMVAD\_SHORT method*), 376

`get_protection()` (*vm\_area\_struct method*), 337

`get_protocol_as_string()` (*socket method*), 353

`get_range_alias()` (*vm\_map\_entry method*), 354

`get_render_options()` (*CLIRenderer method*), 33

`get_render_options()` (*CSVRenderer method*), 33

`get_render_options()` (*PrettyTextRenderer method*), 33

`get_render_options()` (*QuickTextRenderer method*), 33

`get_render_options()` (*Renderer method*), 115

`get_report_hook()` (*PdbRetreiver method*), 398

`get_requirements()` (*AutomagicInterface class method*), 87

`get_requirements()` (*Bash class method*), 205, 222

`get_requirements()` (*BashIntermedSymbols class method*), 341

`get_requirements()` (*BufferDataLayer class method*), 156

`get_requirements()` (*Certificates class method*), 246

`get_requirements()` (*Check\_afinfo class method*), 206

`get_requirements()` (*Check\_syscall class method*), 208, 223, 244

`get_requirements()` (*Check\_sysctl class method*), 225

`get_requirements()` (*Check\_trap\_table class method*), 226

`get_requirements()` (*CmdLine class method*), 256

`get_requirements()` (*ComplexListRequirement class method*), 65

`get_requirements()` (*ConfigurableInterface class method*), 91

`get_requirements()` (*ConfigWriter class method*), 303

`get_requirements()` (*ConstructionMagic class method*), 36

`get_requirements()` (*DataLayerInterface class method*), 103

`get_requirements()` (*DllDump class method*), 258

`get_requirements()` (*DllList class method*), 259

`get_requirements()` (*DriverIrp class method*), 261

`get_requirements()` (*DriverScan class method*), 262

`get_requirements()` (*Elfs class method*), 209

`get_requirements()` (*FileLayer class method*), 158

`get_requirements()` (*FileScan class method*), 264

`get_requirements()` (*Handles class method*), 266

`get_requirements()` (*HiveList class method*), 247

`get_requirements()` (*HiveScan class method*), 249

`get_requirements()` (*Ifconfig class method*), 228

`get_requirements()` (*Info class method*), 268

`get_requirements()` (*Intel class method*), 130

`get_requirements()` (*Intel32e class method*), 132

`get_requirements()` (*IntelPAE class method*), 134

`get_requirements()` (*IntermediateSymbolTable class method*), 403

`get_requirements()` (*ISFormatTable class*



*method*), 399  
 get\_requirements() (*KernelPDBScanner class method*), 46  
 get\_requirements() (*LayerListRequirement class method*), 68  
 get\_requirements() (*LayerStacker class method*), 49  
 get\_requirements() (*LayerWriter class method*), 304  
 get\_requirements() (*LimeLayer class method*), 146  
 get\_requirements() (*LinearlyMappedLayer class method*), 148  
 get\_requirements() (*LinuxBannerCache class method*), 38  
 get\_requirements() (*LinuxKernelIntermedSymbols class method*), 321  
 get\_requirements() (*LinuxSymbolFinder class method*), 40  
 get\_requirements() (*Lsmod class method*), 211, 229  
 get\_requirements() (*Lsof class method*), 213  
 get\_requirements() (*Lsof class method*), 231  
 get\_requirements() (*MacBannerCache class method*), 42  
 get\_requirements() (*MacKernelIntermedSymbols class method*), 343  
 get\_requirements() (*MacSymbolFinder class method*), 43  
 get\_requirements() (*Malfind class method*), 214, 232, 269  
 get\_requirements() (*Maps class method*), 216, 235  
 get\_requirements() (*ModDump class method*), 272  
 get\_requirements() (*ModScan class method*), 273  
 get\_requirements() (*Modules class method*), 275  
 get\_requirements() (*MutantScan class method*), 277  
 get\_requirements() (*Netstat class method*), 234  
 get\_requirements() (*PdbMSFStream class method*), 151  
 get\_requirements() (*PdbMultiStreamFormat class method*), 153  
 get\_requirements() (*PluginInterface class method*), 113  
 get\_requirements() (*PoolScanner class method*), 279  
 get\_requirements() (*PrintKey class method*), 251  
 get\_requirements() (*ProcDump class method*), 283  
 get\_requirements() (*Psaux class method*), 237  
 get\_requirements() (*PsList class method*), 218, 239, 285  
 get\_requirements() (*PsScan class method*), 287  
 get\_requirements() (*PsTree class method*), 219, 240, 289  
 get\_requirements() (*RegistryHive class method*), 161  
 get\_requirements() (*SegmentedLayer class method*), 164  
 get\_requirements() (*SSDT class method*), 291  
 get\_requirements() (*Statistics class method*), 254  
 get\_requirements() (*Strings class method*), 293  
 get\_requirements() (*SymbolBannerCache class method*), 51  
 get\_requirements() (*SymbolFinder class method*), 53  
 get\_requirements() (*SymbolTableInterface class method*), 123  
 get\_requirements() (*SymlinkScan class method*), 294  
 get\_requirements() (*Tasks class method*), 242  
 get\_requirements() (*Timeliner class method*), 306  
 get\_requirements() (*TranslationLayerInterface class method*), 107  
 get\_requirements() (*UserAssist class method*), 253  
 get\_requirements() (*VadDump class method*), 296  
 get\_requirements() (*VadInfo class method*), 297  
 get\_requirements() (*VerInfo class method*), 299  
 get\_requirements() (*Version1Format class method*), 405  
 get\_requirements() (*Version2Format class method*), 407  
 get\_requirements() (*Version3Format class method*), 410  
 get\_requirements() (*Version4Format class method*), 412  
 get\_requirements() (*Version5Format class method*), 415  
 get\_requirements() (*Version6Format class method*), 417  
 get\_requirements() (*Version7Format class method*), 420  
 get\_requirements() (*VirtMap class method*), 301  
 get\_requirements() (*VmwareLayer class method*), 166  
 get\_requirements() (*Volshell class method*), 25, 27, 29, 31  
 get\_requirements() (*WindowsCrash-Dump32Layer class method*), 127  
 get\_requirements() (*WindowsIntel class method*), 136  
 get\_requirements() (*WindowsIntel32e class method*), 139  
 get\_requirements() (*WindowsIntelPAE class method*), 141

`get_requirements()` (*WindowsKernelIntermedSymbols class method*), 359

`get_requirements()` (*WindowsMixin class method*), 143

`get_requirements()` (*WinSwapLayers class method*), 57

`get_requirements()` (*WintelHelper class method*), 59

`get_right_child()` (*MMVAD method*), 374

`get_right_child()` (*MMVAD\_SHORT method*), 376

`get_root_dentry()` (*fs\_struct method*), 325

`get_root_mnt()` (*fs\_struct method*), 325

`get_sections()` (*IMAGE\_NT\_HEADERS method*), 386

`get_session_id()` (*EPROCESS method*), 363

`get_session_layers()` (*ModDump class method*), 272

`get_size_from_index()` (*PdbReader method*), 397

`get_special_path()` (*vm\_map\_entry method*), 354

`get_start()` (*MMVAD method*), 374

`get_start()` (*MMVAD\_SHORT method*), 376

`get_state()` (*socket method*), 353

`get_stream()` (*PdbMultiStreamFormat method*), 153

`get_string()` (*UNICODE\_STRING method*), 381

`get_subkeys()` (*CM\_KEY\_NODE method*), 390

`get_symbol()` (*BaseSymbolTableInterface method*), 118

`get_symbol()` (*BashIntermedSymbols method*), 341

`get_symbol()` (*IntermediateSymbolTable method*), 403

`get_symbol()` (*ISFormatTable method*), 400

`get_symbol()` (*LinuxKernelIntermedSymbols method*), 321

`get_symbol()` (*MacKernelIntermedSymbols method*), 343

`get_symbol()` (*Module method*), 83

`get_symbol()` (*ModuleInterface method*), 101

`get_symbol()` (*NativeTable method*), 422

`get_symbol()` (*NativeTableInterface method*), 120

`get_symbol()` (*SizedModule method*), 85

`get_symbol()` (*SymbolSpace method*), 317

`get_symbol()` (*SymbolSpaceInterface method*), 122

`get_symbol()` (*SymbolTableInterface method*), 123

`get_symbol()` (*Version1Format method*), 405

`get_symbol()` (*Version2Format method*), 407

`get_symbol()` (*Version3Format method*), 410

`get_symbol()` (*Version4Format method*), 412

`get_symbol()` (*Version5Format method*), 415

`get_symbol()` (*Version6Format method*), 417

`get_symbol()` (*Version7Format method*), 420

`get_symbol()` (*WindowsKernelIntermedSymbols method*), 359

`get_symbol_table()` (*AggregateType method*), 169

`get_symbol_table()` (*Array method*), 170

`get_symbol_table()` (*BitField method*), 172

`get_symbol_table()` (*Boolean method*), 174

`get_symbol_table()` (*Bytes method*), 176

`get_symbol_table()` (*Char method*), 180

`get_symbol_table()` (*ClassType method*), 182

`get_symbol_table()` (*CM\_KEY\_BODY method*), 388

`get_symbol_table()` (*CM\_KEY\_NODE method*), 390

`get_symbol_table()` (*CM\_KEY\_VALUE method*), 391

`get_symbol_table()` (*CMHIVE method*), 387

`get_symbol_table()` (*dentry method*), 323

`get_symbol_table()` (*DEVICE\_OBJECT method*), 361

`get_symbol_table()` (*DRIVER\_OBJECT method*), 362

`get_symbol_table()` (*Enumeration method*), 184

`get_symbol_table()` (*EPROCESS method*), 363

`get_symbol_table()` (*ETHREAD method*), 365

`get_symbol_table()` (*EX\_FAST\_REF method*), 366

`get_symbol_table()` (*ExecutiveObject method*), 367

`get_symbol_table()` (*FILE\_OBJECT method*), 369

`get_symbol_table()` (*fileglob method*), 345

`get_symbol_table()` (*files\_struct method*), 324

`get_symbol_table()` (*Float method*), 186

`get_symbol_table()` (*fs\_struct method*), 325

`get_symbol_table()` (*Function method*), 187

`get_symbol_table()` (*GenericIntelProcess method*), 319

`get_symbol_table()` (*hist\_entry method*), 339

`get_symbol_table()` (*HMAP\_ENTRY method*), 392

`get_symbol_table()` (*ifnet method*), 346

`get_symbol_table()` (*IMAGE\_DOS\_HEADER method*), 384

`get_symbol_table()` (*IMAGE\_NT\_HEADERS method*), 386

`get_symbol_table()` (*inpcb method*), 347

`get_symbol_table()` (*Integer method*), 189

`get_symbol_table()` (*KDDEBUGGER\_DATA64 method*), 383

`get_symbol_table()` (*KMUTANT method*), 370

`get_symbol_table()` (*KSYSTEM\_TIME method*), 371

`get_symbol_table()` (*LIST\_ENTRY method*), 372

`get_symbol_table()` (*list\_head method*), 326

`get_symbol_table()` (*mm\_struct method*), 327

`get_symbol_table()` (*MMVAD method*), 374

[get\\_symbol\\_table\(\) \(MMVAD\\_SHORT method\), 376](#)  
[get\\_symbol\\_table\(\) \(module method\), 328](#)  
[get\\_symbol\\_table\(\) \(mount method\), 330](#)  
[get\\_symbol\\_table\(\) \(OBJECT\\_HEADER method\), 377](#)  
[get\\_symbol\\_table\(\) \(OBJECT\\_SYMBOLIC\\_LINK method\), 378](#)  
[get\\_symbol\\_table\(\) \(ObjectInterface method\), 111](#)  
[get\\_symbol\\_table\(\) \(Pointer method\), 191](#)  
[get\\_symbol\\_table\(\) \(POOL\\_HEADER method\), 380](#)  
[get\\_symbol\\_table\(\) \(PrimitiveObject method\), 192](#)  
[get\\_symbol\\_table\(\) \(proc method\), 348](#)  
[get\\_symbol\\_table\(\) \(qstr method\), 331](#)  
[get\\_symbol\\_table\(\) \(queue\\_entry method\), 350](#)  
[get\\_symbol\\_table\(\) \(SERVICE\\_HEADER method\), 394](#)  
[get\\_symbol\\_table\(\) \(SERVICE\\_RECORD method\), 396](#)  
[get\\_symbol\\_table\(\) \(sockaddr method\), 351](#)  
[get\\_symbol\\_table\(\) \(sockaddr\\_dl method\), 352](#)  
[get\\_symbol\\_table\(\) \(socket method\), 353](#)  
[get\\_symbol\\_table\(\) \(String method\), 194](#)  
[get\\_symbol\\_table\(\) \(struct\\_file method\), 332](#)  
[get\\_symbol\\_table\(\) \(StructType method\), 199](#)  
[get\\_symbol\\_table\(\) \(super\\_block method\), 333](#)  
[get\\_symbol\\_table\(\) \(task\\_struct method\), 335](#)  
[get\\_symbol\\_table\(\) \(UNICODE\\_STRING method\), 381](#)  
[get\\_symbol\\_table\(\) \(UnionType method\), 200](#)  
[get\\_symbol\\_table\(\) \(vfsmount method\), 336](#)  
[get\\_symbol\\_table\(\) \(vm\\_area\\_struct method\), 337](#)  
[get\\_symbol\\_table\(\) \(vm\\_map\\_entry method\), 355](#)  
[get\\_symbol\\_table\(\) \(vm\\_map\\_object method\), 356](#)  
[get\\_symbol\\_table\(\) \(vnode method\), 357](#)  
[get\\_symbol\\_table\(\) \(Void method\), 201](#)  
[get\\_symbol\\_type\(\) \(BaseSymbolTableInterface method\), 118](#)  
[get\\_symbol\\_type\(\) \(BashIntermedSymbols method\), 341](#)  
[get\\_symbol\\_type\(\) \(IntermediateSymbolTable method\), 403](#)  
[get\\_symbol\\_type\(\) \(ISFormatTable method\), 400](#)  
[get\\_symbol\\_type\(\) \(LinuxKernelIntermedSymbols method\), 321](#)  
[get\\_symbol\\_type\(\) \(MacKernelIntermedSymbols method\), 343](#)  
[get\\_symbol\\_type\(\) \(NativeTable method\), 422](#)  
[get\\_symbol\\_type\(\) \(NativeTableInterface method\), 120](#)  
[get\\_symbol\\_type\(\) \(SymbolTableInterface method\), 123](#)  
[get\\_symbol\\_type\(\) \(Version1Format method\), 405](#)  
[get\\_symbol\\_type\(\) \(Version2Format method\), 407](#)  
[get\\_symbol\\_type\(\) \(Version3Format method\), 410](#)  
[get\\_symbol\\_type\(\) \(Version4Format method\), 412](#)  
[get\\_symbol\\_type\(\) \(Version5Format method\), 415](#)  
[get\\_symbol\\_type\(\) \(Version6Format method\), 417](#)  
[get\\_symbol\\_type\(\) \(Version7Format method\), 420](#)  
[get\\_symbol\\_type\(\) \(WindowsKernelIntermedSymbols method\), 359](#)  
[get\\_symbols\\_by\\_absolute\\_location\(\) \(SizedModule method\), 85](#)  
[get\\_symbols\\_by\\_location\(\) \(BaseSymbolTableInterface method\), 118](#)  
[get\\_symbols\\_by\\_location\(\) \(BashIntermedSymbols method\), 341](#)  
[get\\_symbols\\_by\\_location\(\) \(IntermediateSymbolTable method\), 403](#)  
[get\\_symbols\\_by\\_location\(\) \(ISFormatTable method\), 400](#)  
[get\\_symbols\\_by\\_location\(\) \(LinuxKernelIntermedSymbols method\), 321](#)  
[get\\_symbols\\_by\\_location\(\) \(MacKernelIntermedSymbols method\), 343](#)  
[get\\_symbols\\_by\\_location\(\) \(NativeTable method\), 422](#)  
[get\\_symbols\\_by\\_location\(\) \(NativeTableInterface method\), 120](#)  
[get\\_symbols\\_by\\_location\(\) \(SymbolSpace method\), 317](#)  
[get\\_symbols\\_by\\_location\(\) \(SymbolSpaceInterface method\), 122](#)  
[get\\_symbols\\_by\\_location\(\) \(SymbolTableInterface method\), 123](#)  
[get\\_symbols\\_by\\_location\(\) \(Version1Format method\), 405](#)  
[get\\_symbols\\_by\\_location\(\) \(Version2Format method\), 408](#)  
[get\\_symbols\\_by\\_location\(\) \(Version3Format method\), 410](#)  
[get\\_symbols\\_by\\_location\(\) \(Version4Format method\), 413](#)  
[get\\_symbols\\_by\\_location\(\) \(Version5Format method\), 415](#)  
[get\\_symbols\\_by\\_location\(\) \(Version6Format method\), 418](#)  
[get\\_symbols\\_by\\_location\(\) \(Version7Format method\), 420](#)  
[get\\_symbols\\_by\\_location\(\) \(WindowsKernelIntermedSymbols method\), 359](#)  
[get\\_symbols\\_by\\_type\(\) \(BaseSymbolTableInterface method\), 118](#)

`get_symbols_by_type()` (*BashIntermedSymbols method*), 341

`get_symbols_by_type()` (*IntermediateSymbolTable method*), 403

`get_symbols_by_type()` (*ISFormatTable method*), 400

`get_symbols_by_type()` (*LinuxKernelIntermedSymbols method*), 321

`get_symbols_by_type()` (*MacKernelIntermedSymbols method*), 343

`get_symbols_by_type()` (*NativeTable method*), 423

`get_symbols_by_type()` (*NativeTableInterface method*), 120

`get_symbols_by_type()` (*SymbolSpace method*), 317

`get_symbols_by_type()` (*SymbolSpaceInterface method*), 122

`get_symbols_by_type()` (*SymbolTableInterface method*), 123

`get_symbols_by_type()` (*Version1Format method*), 405

`get_symbols_by_type()` (*Version2Format method*), 408

`get_symbols_by_type()` (*Version3Format method*), 410

`get_symbols_by_type()` (*Version4Format method*), 413

`get_symbols_by_type()` (*Version5Format method*), 415

`get_symbols_by_type()` (*Version6Format method*), 418

`get_symbols_by_type()` (*Version7Format method*), 420

`get_symbols_by_type()` (*WindowsKernelIntermedSymbols method*), 359

`get_tag(MMVAD attribute)`, 374

`get_tag(MMVAD_SHORT attribute)`, 376

`get_task()` (*proc method*), 349

`get_tcp_state()` (*inpcb method*), 347

`get_time()` (*KSYSTEM\_TIME method*), 371

`get_time_as_integer()` (*hist\_entry method*), 339

`get_time_object()` (*hist\_entry method*), 339

`get_type()` (*BaseSymbolTableInterface method*), 119

`get_type()` (*BashIntermedSymbols method*), 341

`get_type()` (*IntermediateSymbolTable method*), 403

`get_type()` (*ISFormatTable method*), 400

`get_type()` (*LinuxKernelIntermedSymbols method*), 321

`get_type()` (*MacKernelIntermedSymbols method*), 343

`get_type()` (*Module method*), 83

`get_type()` (*ModuleInterface method*), 101

`get_type()` (*NativeTable method*), 423

`get_type()` (*NativeTableInterface method*), 120

`get_type()` (*SERVICE\_RECORD method*), 396

`get_type()` (*SizedModule method*), 85

`get_type()` (*SymbolSpace method*), 317

`get_type()` (*SymbolSpaceInterface method*), 122

`get_type()` (*SymbolTableInterface method*), 124

`get_type()` (*Version1Format method*), 405

`get_type()` (*Version2Format method*), 408

`get_type()` (*Version3Format method*), 410

`get_type()` (*Version4Format method*), 413

`get_type()` (*Version5Format method*), 415

`get_type()` (*Version6Format method*), 418

`get_type()` (*Version7Format method*), 420

`get_type()` (*WindowsKernelIntermedSymbols method*), 359

`get_type_class()` (*BaseSymbolTableInterface method*), 119

`get_type_class()` (*BashIntermedSymbols method*), 341

`get_type_class()` (*IntermediateSymbolTable method*), 403

`get_type_class()` (*ISFormatTable method*), 400

`get_type_class()` (*LinuxKernelIntermedSymbols method*), 321

`get_type_class()` (*MacKernelIntermedSymbols method*), 343

`get_type_class()` (*NativeTable method*), 423

`get_type_class()` (*NativeTableInterface method*), 120

`get_type_class()` (*SymbolTableInterface method*), 124

`get_type_class()` (*Version1Format method*), 405

`get_type_class()` (*Version2Format method*), 408

`get_type_class()` (*Version3Format method*), 410

`get_type_class()` (*Version4Format method*), 413

`get_type_class()` (*Version5Format method*), 415

`get_type_class()` (*Version6Format method*), 418

`get_type_class()` (*Version7Format method*), 420

`get_type_class()` (*WindowsKernelIntermedSymbols method*), 359

`get_type_from_index()` (*PdbReader method*), 397

`get_type_map()` (*Handles class method*), 266

`get_usable_plugins()` (*Timeliner class method*), 306

`get_vad_root()` (*EPROCESS method*), 363

`get_values()` (*CM\_KEY\_NODE method*), 390

`get_version_information()` (*VerInfo class method*), 299

`get_vfsmnt()` (*struct\_file method*), 332

`get_vnode()` (*vm\_map\_entry method*), 355

`get_volatile()` (*CM\_KEY\_NODE method*), 390

`get_wow_64_process()` (*EPROCESS method*), 364

`group_structure` (*VmwareLayer attribute*), 167



## H

handler\_order (*JarHandler* attribute), 163

Handles (class in *volatility.plugins.windows.handles*), 265

handles() (*Handles* method), 266

has\_enumeration() (*Module* method), 83

has\_enumeration() (*ModuleInterface* method), 101

has\_enumeration() (*SizedModule* method), 85

has\_enumeration() (*SymbolSpace* method), 317

has\_enumeration() (*SymbolSpaceInterface* method), 122

has\_member() (*AggregateType* method), 169

has\_member() (*AggregateType.VolTemplateProxy* class method), 169

has\_member() (*Array* method), 170

has\_member() (*Array.VolTemplateProxy* class method), 170

has\_member() (*BitField* method), 172

has\_member() (*BitField.VolTemplateProxy* class method), 171

has\_member() (*Boolean* method), 174

has\_member() (*Boolean.VolTemplateProxy* class method), 173

has\_member() (*Bytes* method), 176

has\_member() (*Bytes.VolTemplateProxy* class method), 175

has\_member() (*Char* method), 180

has\_member() (*Char.VolTemplateProxy* class method), 179

has\_member() (*ClassType* method), 182

has\_member() (*ClassType.VolTemplateProxy* class method), 181

has\_member() (*CM\_KEY\_BODY* method), 388

has\_member() (*CM\_KEY\_BODY.VolTemplateProxy* class method), 388

has\_member() (*CM\_KEY\_NODE* method), 390

has\_member() (*CM\_KEY\_NODE.VolTemplateProxy* class method), 389

has\_member() (*CM\_KEY\_VALUE* method), 391

has\_member() (*CM\_KEY\_VALUE.VolTemplateProxy* class method), 390

has\_member() (*CMHIVE* method), 387

has\_member() (*CMHIVE.VolTemplateProxy* class method), 386

has\_member() (*dentry* method), 323

has\_member() (*dentry.VolTemplateProxy* class method), 322

has\_member() (*DEVICE\_OBJECT* method), 361

has\_member() (*DEVICE\_OBJECT.VolTemplateProxy* class method), 360

has\_member() (*DRIVER\_OBJECT* method), 362

has\_member() (*DRIVER\_OBJECT.VolTemplateProxy* class method), 361

has\_member() (*Enumeration* method), 184

has\_member() (*Enumeration.VolTemplateProxy* class method), 183

has\_member() (*EPROCESS* method), 364

has\_member() (*EPROCESS.VolTemplateProxy* class method), 363

has\_member() (*ETHREAD* method), 365

has\_member() (*ETHREAD.VolTemplateProxy* class method), 364

has\_member() (*EX\_FAST\_REF* method), 366

has\_member() (*EX\_FAST\_REF.VolTemplateProxy* class method), 366

has\_member() (*ExecutiveObject* method), 367

has\_member() (*ExecutiveObject.VolTemplateProxy* class method), 367

has\_member() (*FILE\_OBJECT* method), 369

has\_member() (*FILE\_OBJECT.VolTemplateProxy* class method), 368

has\_member() (*fileglob* method), 345

has\_member() (*fileglob.VolTemplateProxy* class method), 344

has\_member() (*files\_struct* method), 324

has\_member() (*files\_struct.VolTemplateProxy* class method), 323

has\_member() (*Float* method), 186

has\_member() (*Float.VolTemplateProxy* class method), 185

has\_member() (*fs\_struct* method), 325

has\_member() (*fs\_struct.VolTemplateProxy* class method), 324

has\_member() (*Function* method), 187

has\_member() (*Function.VolTemplateProxy* class method), 187

has\_member() (*GenericIntelProcess* method), 319

has\_member() (*GenericIntelProcess.VolTemplateProxy* class method), 318

has\_member() (*hist\_entry* method), 339

has\_member() (*hist\_entry.VolTemplateProxy* class method), 338

has\_member() (*HMAP\_ENTRY* method), 392

has\_member() (*HMAP\_ENTRY.VolTemplateProxy* class method), 392

has\_member() (*ifnet* method), 346

has\_member() (*ifnet.VolTemplateProxy* class method), 346

has\_member() (*IMAGE\_DOS\_HEADER* method), 384

has\_member() (*IMAGE\_DOS\_HEADER.VolTemplateProxy* class method), 383

has\_member() (*IMAGE\_NT\_HEADERS* method), 386

has\_member() (*IMAGE\_NT\_HEADERS.VolTemplateProxy* class method), 385

has\_member() (*inpcb* method), 347

has\_member() (*inpcb.VolTemplateProxy* class method), 347

`has_member()` (*Integer* method), 189  
`has_member()` (*Integer.VolTemplateProxy* class method), 188  
`has_member()` (*KDDEBUGGER\_DATA64* method), 383  
`has_member()` (*KDDEBUGGER\_DATA64.VolTemplateProxy* class method), 382  
`has_member()` (*KMUTANT* method), 370  
`has_member()` (*KMUTANT.VolTemplateProxy* class method), 370  
`has_member()` (*KSYSTEM\_TIME* method), 371  
`has_member()` (*KSYSTEM\_TIME.VolTemplateProxy* class method), 371  
`has_member()` (*LIST\_ENTRY* method), 373  
`has_member()` (*LIST\_ENTRY.VolTemplateProxy* class method), 372  
`has_member()` (*list\_head* method), 326  
`has_member()` (*list\_head.VolTemplateProxy* class method), 326  
`has_member()` (*mm\_struct* method), 327  
`has_member()` (*mm\_struct.VolTemplateProxy* class method), 327  
`has_member()` (*MMVAD* method), 374  
`has_member()` (*MMVAD.VolTemplateProxy* class method), 373  
`has_member()` (*MMVAD\_SHORT* method), 376  
`has_member()` (*MMVAD\_SHORT.VolTemplateProxy* class method), 375  
`has_member()` (*module* method), 329  
`has_member()` (*module.VolTemplateProxy* class method), 328  
`has_member()` (*mount* method), 330  
`has_member()` (*mount.VolTemplateProxy* class method), 329  
`has_member()` (*OBJECT\_HEADER* method), 377  
`has_member()` (*OBJECT\_HEADER.VolTemplateProxy* class method), 377  
`has_member()` (*OBJECT\_SYMBOLIC\_LINK* method), 379  
`has_member()` (*OBJECT\_SYMBOLIC\_LINK.VolTemplateProxy* class method), 378  
`has_member()` (*ObjectInterface* method), 111  
`has_member()` (*ObjectInterface.VolTemplateProxy* class method), 110  
`has_member()` (*ObjectTemplate* method), 202  
`has_member()` (*Pointer* method), 191  
`has_member()` (*Pointer.VolTemplateProxy* class method), 190  
`has_member()` (*POOL\_HEADER* method), 380  
`has_member()` (*POOL\_HEADER.VolTemplateProxy* class method), 379  
`has_member()` (*PrimitiveObject* method), 192  
`has_member()` (*PrimitiveObject.VolTemplateProxy* class method), 192  
`has_member()` (*proc* method), 349  
`has_member()` (*proc.VolTemplateProxy* class method), 348  
`has_member()` (*qstr* method), 331  
`has_member()` (*qstr.VolTemplateProxy* class method), 330  
`has_member()` (*queue\_entry* method), 350  
`has_member()` (*queue\_entry.VolTemplateProxy* class method), 349  
`has_member()` (*ReferenceTemplate* method), 203  
`has_member()` (*SERVICE\_HEADER* method), 394  
`has_member()` (*SERVICE\_HEADER.VolTemplateProxy* class method), 394  
`has_member()` (*SERVICE\_RECORD* method), 396  
`has_member()` (*SERVICE\_RECORD.VolTemplateProxy* class method), 395  
`has_member()` (*sockaddr* method), 351  
`has_member()` (*sockaddr.VolTemplateProxy* class method), 350  
`has_member()` (*sockaddr\_dl* method), 352  
`has_member()` (*sockaddr\_dl.VolTemplateProxy* class method), 352  
`has_member()` (*socket* method), 353  
`has_member()` (*socket.VolTemplateProxy* class method), 353  
`has_member()` (*String* method), 194  
`has_member()` (*String.VolTemplateProxy* class method), 193  
`has_member()` (*struct\_file* method), 332  
`has_member()` (*struct\_file.VolTemplateProxy* class method), 332  
`has_member()` (*StructType* method), 199  
`has_member()` (*StructType.VolTemplateProxy* class method), 198  
`has_member()` (*super\_block* method), 333  
`has_member()` (*super\_block.VolTemplateProxy* class method), 333  
`has_member()` (*SymbolSpace.UnresolvedTemplate* method), 316  
`has_member()` (*task\_struct* method), 335  
`has_member()` (*task\_struct.VolTemplateProxy* class method), 334  
`has_member()` (*Template* method), 112  
`has_member()` (*UNICODE\_STRING* method), 381  
`has_member()` (*UNICODE\_STRING.VolTemplateProxy* class method), 381  
`has_member()` (*UnionType* method), 200  
`has_member()` (*UnionType.VolTemplateProxy* class method), 200

- method*), 199
- `has_member()` (*vfsmount method*), 336
- `has_member()` (*vfsmount.VolTemplateProxy class method*), 335
- `has_member()` (*vm\_area\_struct method*), 337
- `has_member()` (*vm\_area\_struct.VolTemplateProxy class method*), 337
- `has_member()` (*vm\_map\_entry method*), 355
- `has_member()` (*vm\_map\_entry.VolTemplateProxy class method*), 354
- `has_member()` (*vm\_map\_object method*), 356
- `has_member()` (*vm\_map\_object.VolTemplateProxy class method*), 355
- `has_member()` (*vnnode method*), 357
- `has_member()` (*vnnode.VolTemplateProxy class method*), 356
- `has_member()` (*Void method*), 201
- `has_member()` (*Void.VolTemplateProxy class method*), 200
- `has_symbol()` (*Module method*), 83
- `has_symbol()` (*ModuleInterface method*), 101
- `has_symbol()` (*SizedModule method*), 85
- `has_symbol()` (*SymbolSpace method*), 317
- `has_symbol()` (*SymbolSpaceInterface method*), 122
- `has_type()` (*Module method*), 83
- `has_type()` (*ModuleInterface method*), 101
- `has_type()` (*SizedModule method*), 85
- `has_type()` (*SymbolSpace method*), 317
- `has_type()` (*SymbolSpaceInterface method*), 122
- `hash()` (*SizedModule property*), 85
- `header_structure` (*VmwareLayer attribute*), 167
- `headerpages` (*WindowsCrashDump32Layer attribute*), 127
- `help()` (*Volshell method*), 25, 27, 29, 32
- `HelpfulSubparserAction` (*class in volatility.cli*), 22
- `Hex` (*class in volatility.framework.renderers.format\_hints*), 311
- `hex()` (*Bytes method*), 176
- `hex()` (*Float method*), 186
- `hex()` (*HexBytes method*), 313
- `hex_bytes_as_text()` (*in module volatility.cli.text\_renderer*), 34
- `HexBytes` (*class in volatility.framework.renderers.format\_hints*), 312
- `hide_from_subclasses()` (*in module volatility.framework*), 34
- `HierarchicalDict` (*class in volatility.framework.interfaces.configuration*), 95
- `hist_entry` (*class in volatility.framework.symbols.linux.extensions.bash*), 338
- `hist_entry.VolTemplateProxy` (*class in volatility.framework.symbols.linux.extensions.bash*), 338
- `hive_offset()` (*RegistryHive property*), 161
- `HiveList` (*class in volatility.plugins.windows.registry.hivelist*), 247
- `HiveScan` (*class in volatility.plugins.windows.registry.hivescan*), 249
- `HMAP_ENTRY` (*class in volatility.framework.symbols.windows.extensions.registry*), 391
- `HMAP_ENTRY.VolTemplateProxy` (*class in volatility.framework.symbols.windows.extensions.registry*), 392
- I**
- `Ifconfig` (*class in volatility.plugins.mac.ifconfig*), 227
- `ifnet` (*class in volatility.framework.symbols.mac.extensions*), 345
- `ifnet.VolTemplateProxy` (*class in volatility.framework.symbols.mac.extensions*), 345
- `imag` (*Bin attribute*), 311
- `imag` (*BitField attribute*), 172
- `imag` (*Boolean attribute*), 174
- `imag` (*Char attribute*), 180
- `imag` (*Enumeration attribute*), 184
- `imag` (*Float attribute*), 186
- `imag` (*Hex attribute*), 312
- `imag` (*Integer attribute*), 189
- `imag` (*Pointer attribute*), 191
- `IMAGE_DOS_HEADER` (*class in volatility.framework.symbols.windows.extensions.pe*), 383
- `IMAGE_DOS_HEADER.VolTemplateProxy` (*class in volatility.framework.symbols.windows.extensions.pe*), 383
- `IMAGE_NT_HEADERS` (*class in volatility.framework.symbols.windows.extensions.pe*), 385
- `IMAGE_NT_HEADERS.VolTemplateProxy` (*class in volatility.framework.symbols.windows.extensions.pe*), 385
- `import_files()` (*in module volatility.framework*), 34
- `index()` (*Array method*), 170
- `index()` (*Bytes method*), 176
- `index()` (*Column method*), 115
- `index()` (*DataFormatInfo method*), 182
- `index()` (*HexBytes method*), 313
- `index()` (*String method*), 195
- `index()` (*TreeNode method*), 117, 309
- `Info` (*class in volatility.plugins.windows.info*), 267
- `inpcb` (*class in volatility.framework.symbols.mac.extensions*), 346

`inpcb.VolTemplateProxy` (class in `volatility.framework.symbols.mac.extensions`), 347

`instance_type` (*BooleanRequirement* attribute), 60

`instance_type` (*BytesRequirement* attribute), 62

`instance_type` (*IntRequirement* attribute), 66

`instance_type` (*SimpleTypeRequirement* attribute), 98

`instance_type` (*StringRequirement* attribute), 74

`instance_type` (*URIRequirement* attribute), 79

`Integer` (class in `volatility.framework.objects`), 188

`Integer.VolTemplateProxy` (class in `volatility.framework.objects`), 188

`Intel` (class in `volatility.framework.layers.intel`), 129

`Intel32e` (class in `volatility.framework.layers.intel`), 131

`IntelPAE` (class in `volatility.framework.layers.intel`), 133

`interface_version()` (in module `volatility.framework`), 34

`IntermediateSymbolTable` (class in `volatility.framework.symbols.intermed`), 401

`IntRequirement` (class in `volatility.framework.configuration.requirements`), 65

`InvalidAddressException`, 424

`invalidate_caches()` (*WarningFindSpec* method), 21

`is_ancestor()` (*TreeGrid* method), 116, 308

`is_integer()` (*Float* method), 186

`is_readable()` (*Pointer* method), 191

`is_suspicious()` (*vm\_area\_struct* method), 337

`is_suspicious()` (*vm\_map\_entry* method), 355

`is_vad_empty()` (*Malfind* class method), 269

`is_valid()` (*BufferDataLayer* method), 156

`is_valid()` (*DataLayerInterface* method), 103

`is_valid()` (*DRIVER\_OBJECT* method), 362

`is_valid()` (*EPROCESS* method), 364

`is_valid()` (*FILE\_OBJECT* method), 369

`is_valid()` (*FileLayer* method), 158

`is_valid()` (*hist\_entry* method), 339

`is_valid()` (*Intel* method), 130

`is_valid()` (*Intel32e* method), 132

`is_valid()` (*IntelPAE* method), 134

`is_valid()` (*KMUTANT* method), 370

`is_valid()` (*LimeLayer* method), 146

`is_valid()` (*LinearlyMappedLayer* method), 148

`is_valid()` (*OBJECT\_HEADER* method), 377

`is_valid()` (*OBJECT\_SYMBOLIC\_LINK* method), 379

`is_valid()` (*PdbMSFStream* method), 151

`is_valid()` (*PdbMultiStreamFormat* method), 153

`is_valid()` (*RegistryHive* method), 161

`is_valid()` (*SegmentedLayer* method), 164

`is_valid()` (*SERVICE\_HEADER* method), 394

`is_valid()` (*SERVICE\_RECORD* method), 396

`is_valid()` (*TranslationLayerInterface* method), 108

`is_valid()` (*vfsmount* method), 336

`is_valid()` (*VmwareLayer* method), 167

`is_valid()` (*WindowsCrashDump32Layer* method), 127

`is_valid()` (*WindowsIntel* method), 136

`is_valid()` (*WindowsIntel32e* method), 139

`is_valid()` (*WindowsIntelPAE* method), 141

`is_valid()` (*WindowsMixin* method), 143

`is_windows_10()` (*PoolScanner* method), 280

`is_windows_7()` (*PoolScanner* method), 280

`is_windows_8_or_later()` (*PoolScanner* method), 280

`isalnum()` (*Bytes* method), 176

`isalnum()` (*HexBytes* method), 313

`isalnum()` (*String* method), 195

`isalpha()` (*Bytes* method), 176

`isalpha()` (*HexBytes* method), 313

`isalpha()` (*String* method), 195

`isascii()` (*Bytes* method), 176

`isascii()` (*HexBytes* method), 313

`isascii()` (*String* method), 195

`isdecimal()` (*String* method), 195

`isdigit()` (*Bytes* method), 176

`isdigit()` (*HexBytes* method), 313

`isdigit()` (*String* method), 195

`ISFormatTable` (class in `volatility.framework.symbols.intermed`), 399

`isidentifier()` (*String* method), 195

`islower()` (*Bytes* method), 176

`islower()` (*HexBytes* method), 313

`islower()` (*String* method), 195

`isnumeric()` (*String* method), 195

`isprintable()` (*String* method), 195

`isspace()` (*Bytes* method), 177

`isspace()` (*HexBytes* method), 313

`isspace()` (*String* method), 195

`istitle()` (*Bytes* method), 177

`istitle()` (*HexBytes* method), 313

`istitle()` (*String* method), 196

`isupper()` (*Bytes* method), 177

`isupper()` (*HexBytes* method), 314

`isupper()` (*String* method), 196

`items()` (*HierarchicalDict* method), 95

`items()` (*LayerContainer* method), 106

`items()` (*ObjectInformation* method), 110

`items()` (*ReadOnlyMapping* method), 111

`items()` (*SymbolSpace* method), 317

`items()` (*SymbolSpaceInterface* method), 122

## J

`JarHandler` (class in `volatility.framework.layers.resources`), 163



`join()` (*Bytes method*), 177  
`join()` (*HexBytes method*), 314  
`join()` (*String method*), 196

## K

`KDDEBUGGER_DATA64` (class in *volatility.framework.symbols.windows.extensions.kdbg*), 382  
`KDDEBUGGER_DATA64.VolTemplateProxy` (class in *volatility.framework.symbols.windows.extensions.kdbg*), 382  
`KERNEL_MODULE_NAMES` (in module *volatility.framework.constants.windows*), 81  
`KernelPDBScanner` (class in *volatility.framework.automagic.pdbscan*), 44  
`KEY_COMP_NAME` (*RegKeyFlags attribute*), 393  
`KEY_HIVE_ENTRY` (*RegKeyFlags attribute*), 393  
`KEY_HIVE_EXIT` (*RegKeyFlags attribute*), 393  
`KEY_IS_VOLATILE` (*RegKeyFlags attribute*), 393  
`key_iterator()` (*PrintKey class method*), 251  
`KEY_NO_DELETE` (*RegKeyFlags attribute*), 393  
`KEY_PREFEF_HANDLE` (*RegKeyFlags attribute*), 393  
`KEY_SYM_LINK` (*RegKeyFlags attribute*), 393  
`KEY_VIRT_MIRRORED` (*RegKeyFlags attribute*), 393  
`KEY_VIRT_TARGET` (*RegKeyFlags attribute*), 393  
`KEY_VIRTUAL_STORE` (*RegKeyFlags attribute*), 393  
`keys()` (*HierarchicalDict method*), 95  
`keys()` (*LayerContainer method*), 106  
`keys()` (*ObjectInformation method*), 110  
`keys()` (*ReadOnlyMapping method*), 111  
`keys()` (*SymbolSpace method*), 317  
`keys()` (*SymbolSpaceInterface method*), 122  
`KMUTANT` (class in *volatility.framework.symbols.windows.extensions*), 369  
`KMUTANT.VolTemplateProxy` (class in *volatility.framework.symbols.windows.extensions*), 369  
`KSYSTEM_TIME` (class in *volatility.framework.symbols.windows.extensions*), 371  
`KSYSTEM_TIME.VolTemplateProxy` (class in *volatility.framework.symbols.windows.extensions*), 371

## L

`layer_name()` (*BytesScanner property*), 125  
`layer_name()` (*Module property*), 83  
`layer_name()` (*ModuleInterface property*), 102  
`layer_name()` (*MultiStringScanner property*), 125  
`layer_name()` (*PageMapScanner property*), 56  
`layer_name()` (*PdbSignatureScanner property*), 48

`layer_name()` (*PoolHeaderScanner property*), 278  
`layer_name()` (*RegExScanner property*), 125  
`layer_name()` (*ScannerInterface property*), 107  
`layer_name()` (*SizedModule property*), 85  
`LayerContainer` (class in *volatility.framework.interfaces.layers*), 105  
`LayerException`, 424  
`LayerListRequirement` (class in *volatility.framework.configuration.requirements*), 67  
`layers()` (*Context property*), 81  
`layers()` (*ContextInterface property*), 100  
`LayerStacker` (class in *volatility.framework.automagic.stacker*), 48  
`LayerWriter` (class in *volatility.plugins.layerwriter*), 304  
`length()` (*DataFormatInfo property*), 182  
`LimeFormatException`, 145  
`LimeLayer` (class in *volatility.framework.layers.lime*), 145  
`LimeStacker` (class in *volatility.framework.layers.lime*), 147  
`LinearlyMappedLayer` (class in *volatility.framework.layers.linear*), 148  
`LintelStacker` (class in *volatility.framework.automagic.linux*), 37  
`LINUX_BANNERS_PATH` (in module *volatility.framework.constants*), 80  
`LinuxBannerCache` (class in *volatility.framework.automagic.linux*), 37  
`LinuxKernelIntermedSymbols` (class in *volatility.framework.symbols.linux*), 319  
`LinuxMetadata` (class in *volatility.framework.symbols.metadata*), 421  
`LinuxSymbolFinder` (class in *volatility.framework.automagic.linux*), 39  
`LinuxUtilities` (class in *volatility.framework.automagic.linux*), 40  
`LIST_ENTRY` (class in *volatility.framework.symbols.windows.extensions*), 372  
`LIST_ENTRY.VolTemplateProxy` (class in *volatility.framework.symbols.windows.extensions*), 372  
`list_head` (class in *volatility.framework.symbols.linux.extensions*), 325  
`list_head.VolTemplateProxy` (class in *volatility.framework.symbols.linux.extensions*), 325  
`list_hive_objects()` (*HiveList class method*), 247  
`list_hives()` (*HiveList class method*), 248  
`list_injections()` (*Malfind class method*), 270  
`list_modules()` (*Lsmmod class method*), 211, 229  
`list_modules()` (*Modules class method*), 275

`list_plugins()` (in module `volatility.framework`), 34  
`list_processes()` (*PsList* class method), 285  
`list_processes()` (*PsTree* class method), 289  
`list_processes()` (*Volshell* method), 32  
`list_tasks()` (*PsList* class method), 218, 239  
`list_tasks()` (*PsTree* class method), 220  
`list_tasks()` (*Tasks* class method), 242  
`list_tasks()` (*Volshell* method), 27, 29  
`list_userassist()` (*UserAssist* method), 253  
`list_vads()` (*VadInfo* class method), 297  
`ListRequirement` (class in `volatility.framework.configuration.requirements`), 69  
`ljust()` (*Bytes* method), 177  
`ljust()` (*HexBytes* method), 314  
`ljust()` (*String* method), 196  
`load_banners()` (*LinuxBannerCache* class method), 38  
`load_banners()` (*MacBannerCache* class method), 42  
`load_banners()` (*SymbolBannerCache* class method), 51  
`load_cached_validations()` (in module `volatility.schemas`), 425  
`load_order_modules()` (*EPROCESS* method), 364  
`load_pdb_layer()` (*PdbReader* class method), 397  
`location()` (*FileLayer* property), 158  
`LOGLEVEL_V` (in module `volatility.framework.constants`), 80  
`LOGLEVEL_VV` (in module `volatility.framework.constants`), 80  
`LOGLEVEL_VVV` (in module `volatility.framework.constants`), 80  
`LOGLEVEL_VVVV` (in module `volatility.framework.constants`), 80  
`lookup()` (*Enumeration* method), 184  
`lookup()` (*Enumeration.VolTemplateProxy* class method), 183  
`lower()` (*Bytes* method), 177  
`lower()` (*HexBytes* method), 314  
`lower()` (*String* method), 196  
`Lsmode` (class in `volatility.plugins.linux.lsmode`), 210  
`Lsmode` (class in `volatility.plugins.mac.lsmode`), 229  
`Lsof` (class in `volatility.plugins.linux.lsof`), 212  
`Lsof` (class in `volatility.plugins.mac.lsof`), 230  
`lstrip()` (*Bytes* method), 177  
`lstrip()` (*HexBytes* method), 314  
`lstrip()` (*String* method), 196

## M

`MAC_BANNERS_PATH` (in module `volatility.framework.constants`), 80  
`MacBannerCache` (class in `volatility.framework.automagic.mac`), 41  
`MacintelStacker` (class in `volatility.framework.automagic.mac`), 44  
`MacKernelIntermedSymbols` (class in `volatility.framework.symbols.mac`), 342  
`MacSymbolFinder` (class in `volatility.framework.automagic.mac`), 42  
`MacUtilities` (class in `volatility.framework.automagic.mac`), 44  
`MAGIC` (*LimeLayer* attribute), 145  
`main()` (in module `volatility.cli`), 22  
`main()` (in module `volatility.cli.volshell`), 23  
`major()` (*super\_block* property), 333  
`make_subconfig()` (*AutomagicInterface* class method), 88  
`make_subconfig()` (*Bash* class method), 205, 222  
`make_subconfig()` (*BashIntermedSymbols* class method), 341  
`make_subconfig()` (*BufferDataLayer* class method), 156  
`make_subconfig()` (*Certificates* class method), 246  
`make_subconfig()` (*Check\_afinfo* class method), 206  
`make_subconfig()` (*Check\_syscall* class method), 208, 223, 244  
`make_subconfig()` (*Check\_sysctl* class method), 225  
`make_subconfig()` (*Check\_trap\_table* class method), 226  
`make_subconfig()` (*CmdLine* class method), 256  
`make_subconfig()` (*ConfigurableInterface* class method), 91  
`make_subconfig()` (*ConfigWriter* class method), 303  
`make_subconfig()` (*ConstructionMagic* class method), 37  
`make_subconfig()` (*DataLayerInterface* class method), 104  
`make_subconfig()` (*DlIDump* class method), 258  
`make_subconfig()` (*DllList* class method), 259  
`make_subconfig()` (*DriverIrp* class method), 261  
`make_subconfig()` (*DriverScan* class method), 262  
`make_subconfig()` (*Elfs* class method), 210  
`make_subconfig()` (*FileLayer* class method), 158  
`make_subconfig()` (*FileScan* class method), 264  
`make_subconfig()` (*Handles* class method), 266  
`make_subconfig()` (*HiveList* class method), 248  
`make_subconfig()` (*HiveScan* class method), 249  
`make_subconfig()` (*Ifconfig* class method), 228  
`make_subconfig()` (*Info* class method), 268  
`make_subconfig()` (*Intel* class method), 130  
`make_subconfig()` (*Intel32e* class method), 132  
`make_subconfig()` (*IntelPAE* class method), 134  
`make_subconfig()` (*IntermediateSymbolTable* class method), 403  
`make_subconfig()` (*ISFormatTable* class method),

- 400
- `make_subconfig()` (*KernelPDBScanner class method*), 46
- `make_subconfig()` (*LayerStacker class method*), 50
- `make_subconfig()` (*LayerWriter class method*), 304
- `make_subconfig()` (*LimeLayer class method*), 146
- `make_subconfig()` (*LinearlyMappedLayer class method*), 149
- `make_subconfig()` (*LinuxBannerCache class method*), 38
- `make_subconfig()` (*LinuxKernelIntermedSymbols class method*), 321
- `make_subconfig()` (*LinuxSymbolFinder class method*), 40
- `make_subconfig()` (*Lsmode class method*), 211, 229
- `make_subconfig()` (*Lsof class method*), 213
- `make_subconfig()` (*Lsof class method*), 231
- `make_subconfig()` (*MacBannerCache class method*), 42
- `make_subconfig()` (*MacKernelIntermedSymbols class method*), 343
- `make_subconfig()` (*MacSymbolFinder class method*), 43
- `make_subconfig()` (*Malfind class method*), 214, 232, 270
- `make_subconfig()` (*Maps class method*), 216, 235
- `make_subconfig()` (*ModDump class method*), 272
- `make_subconfig()` (*ModScan class method*), 273
- `make_subconfig()` (*Modules class method*), 275
- `make_subconfig()` (*MutantScan class method*), 277
- `make_subconfig()` (*Netstat class method*), 234
- `make_subconfig()` (*PdbMSFStream class method*), 151
- `make_subconfig()` (*PdbMultiStreamFormat class method*), 154
- `make_subconfig()` (*PluginInterface class method*), 114
- `make_subconfig()` (*PoolScanner class method*), 280
- `make_subconfig()` (*PrintKey class method*), 251
- `make_subconfig()` (*ProcDump class method*), 283
- `make_subconfig()` (*Psaux class method*), 237
- `make_subconfig()` (*PsList class method*), 218, 239, 285
- `make_subconfig()` (*PsScan class method*), 287
- `make_subconfig()` (*PsTree class method*), 220, 240, 289
- `make_subconfig()` (*RegistryHive class method*), 161
- `make_subconfig()` (*SegmentedLayer class method*), 164
- `make_subconfig()` (*SSDT class method*), 291
- `make_subconfig()` (*Statistics class method*), 255
- `make_subconfig()` (*Strings class method*), 293
- `make_subconfig()` (*SymbolBannerCache class method*), 51
- `make_subconfig()` (*SymbolFinder class method*), 53
- `make_subconfig()` (*SymbolTableInterface class method*), 124
- `make_subconfig()` (*SymlinkScan class method*), 294
- `make_subconfig()` (*Tasks class method*), 242
- `make_subconfig()` (*Timeliner class method*), 306
- `make_subconfig()` (*TranslationLayerInterface class method*), 108
- `make_subconfig()` (*UserAssist class method*), 253
- `make_subconfig()` (*VadDump class method*), 296
- `make_subconfig()` (*VadInfo class method*), 298
- `make_subconfig()` (*VerInfo class method*), 300
- `make_subconfig()` (*Version1Format class method*), 405
- `make_subconfig()` (*Version2Format class method*), 408
- `make_subconfig()` (*Version3Format class method*), 410
- `make_subconfig()` (*Version4Format class method*), 413
- `make_subconfig()` (*Version5Format class method*), 415
- `make_subconfig()` (*Version6Format class method*), 418
- `make_subconfig()` (*Version7Format class method*), 420
- `make_subconfig()` (*VirtMap class method*), 301
- `make_subconfig()` (*VmwareLayer class method*), 167
- `make_subconfig()` (*Volshell class method*), 25, 27, 29, 32
- `make_subconfig()` (*WindowsCrashDump32Layer class method*), 127
- `make_subconfig()` (*WindowsIntel class method*), 136
- `make_subconfig()` (*WindowsIntel32e class method*), 139
- `make_subconfig()` (*WindowsIntelPAE class method*), 141
- `make_subconfig()` (*WindowsKernelIntermedSymbols class method*), 359
- `make_subconfig()` (*WindowsMixin class method*), 143
- `make_subconfig()` (*WinSwapLayers class method*), 57
- `make_subconfig()` (*WintelHelper class method*), 59
- `maketrans()` (*Bytes static method*), 177
- `maketrans()` (*HexBytes static method*), 314
- `maketrans()` (*String static method*), 196
- `Malfind` (*class in volatility.plugins.linux.malfind*), 214
- `Malfind` (*class in volatility.plugins.mac.malfind*), 232
- `Malfind` (*class in volatility.plugins.windows.malfind*), 269
- `mapping()` (*Intel method*), 130

`mapping()` (*Intel32e method*), 132  
`mapping()` (*IntelPAE method*), 134  
`mapping()` (*LimeLayer method*), 146  
`mapping()` (*LinearlyMappedLayer method*), 149  
`mapping()` (*PdbMSFStream method*), 151  
`mapping()` (*PdbMultiStreamFormat method*), 154  
`mapping()` (*RegistryHive method*), 161  
`mapping()` (*SegmentedLayer method*), 165  
`mapping()` (*TranslationLayerInterface method*), 108  
`mapping()` (*VmwareLayer method*), 167  
`mapping()` (*WindowsCrashDump32Layer method*), 127  
`mapping()` (*WindowsIntel method*), 137  
`mapping()` (*WindowsIntel32e method*), 139  
`mapping()` (*WindowsIntelPAE method*), 141  
`mapping()` (*WindowsMixin method*), 144  
`Maps` (*class in volatility.plugins.linux.proc*), 215  
`Maps` (*class in volatility.plugins.mac.proc\_maps*), 235  
`mask_symbol_table()` (*in module volatility.framework.symbols*), 318  
`max_depth()` (*TreeGrid method*), 116, 308  
`max_pdb_size` (*KernelPDBScanner attribute*), 47  
`maximum_address` (*Intel attribute*), 130  
`maximum_address` (*Intel32e attribute*), 132  
`maximum_address` (*IntelPAE attribute*), 135  
`maximum_address` (*WindowsIntel attribute*), 137  
`maximum_address` (*WindowsIntel32e attribute*), 139  
`maximum_address` (*WindowsIntelPAE attribute*), 141  
`maximum_address` (*WindowsMixin attribute*), 144  
`maximum_address()` (*BufferDataLayer property*), 156  
`maximum_address()` (*DataLayerInterface property*), 104  
`maximum_address()` (*FileLayer property*), 159  
`maximum_address()` (*LimeLayer property*), 146  
`maximum_address()` (*LinearlyMappedLayer property*), 149  
`maximum_address()` (*PdbMSFStream property*), 151  
`maximum_address()` (*PdbMultiStreamFormat property*), 154  
`maximum_address()` (*RegistryHive property*), 161  
`maximum_address()` (*SegmentedLayer property*), 165  
`maximum_address()` (*TranslationLayerInterface property*), 108  
`maximum_address()` (*VmwareLayer property*), 167  
`maximum_address()` (*WindowsCrashDump32Layer property*), 127  
`member()` (*AggregateType method*), 169  
`member()` (*ClassType method*), 182  
`member()` (*CM\_KEY\_BODY method*), 388  
`member()` (*CM\_KEY\_NODE method*), 390  
`member()` (*CM\_KEY\_VALUE method*), 391  
`member()` (*CMHIVE method*), 387  
`member()` (*dentry method*), 323  
`member()` (*DEVICE\_OBJECT method*), 361  
`member()` (*DRIVER\_OBJECT method*), 362  
`member()` (*EPROCESS method*), 364  
`member()` (*ETHREAD method*), 365  
`member()` (*EX\_FAST\_REF method*), 366  
`member()` (*FILE\_OBJECT method*), 369  
`member()` (*fileglob method*), 345  
`member()` (*files\_struct method*), 324  
`member()` (*fs\_struct method*), 325  
`member()` (*GenericIntelProcess method*), 319  
`member()` (*hist\_entry method*), 339  
`member()` (*HMAP\_ENTRY method*), 392  
`member()` (*ifnet method*), 346  
`member()` (*IMAGE\_DOS\_HEADER method*), 384  
`member()` (*IMAGE\_NT\_HEADERS method*), 386  
`member()` (*inpcb method*), 347  
`member()` (*KDDEBUGGER\_DATA64 method*), 383  
`member()` (*KMUTANT method*), 370  
`member()` (*KSYSTEM\_TIME method*), 371  
`member()` (*LIST\_ENTRY method*), 373  
`member()` (*list\_head method*), 326  
`member()` (*mm\_struct method*), 327  
`member()` (*MMVAD method*), 374  
`member()` (*MMVAD\_SHORT method*), 376  
`member()` (*module method*), 329  
`member()` (*mount method*), 330  
`member()` (*OBJECT\_HEADER method*), 377  
`member()` (*OBJECT\_SYMBOLIC\_LINK method*), 379  
`member()` (*POOL\_HEADER method*), 380  
`member()` (*proc method*), 349  
`member()` (*qstr method*), 331  
`member()` (*queue\_entry method*), 350  
`member()` (*SERVICE\_HEADER method*), 395  
`member()` (*SERVICE\_RECORD method*), 396  
`member()` (*sockaddr method*), 351  
`member()` (*sockaddr\_dl method*), 352  
`member()` (*socket method*), 353  
`member()` (*struct\_file method*), 332  
`member()` (*StructType method*), 199  
`member()` (*super\_block method*), 333  
`member()` (*task\_struct method*), 335  
`member()` (*UNICODE\_STRING method*), 381  
`member()` (*UnionType method*), 200  
`member()` (*vfsmount method*), 336  
`member()` (*vm\_area\_struct method*), 337  
`member()` (*vm\_map\_entry method*), 355  
`member()` (*vm\_map\_object method*), 356  
`member()` (*vnode method*), 357  
`merge()` (*HierarchicalDict method*), 95  
`metadata()` (*BashIntermedSymbols property*), 341  
`metadata()` (*BufferDataLayer property*), 156  
`metadata()` (*DataLayerInterface property*), 104  
`metadata()` (*FileLayer property*), 159



- `metadata()` (*Intel property*), 130
- `metadata()` (*Intel32e property*), 132
- `metadata()` (*IntelPAE property*), 135
- `metadata()` (*IntermediateSymbolTable property*), 403
- `metadata()` (*ISFormatTable property*), 400
- `metadata()` (*LimeLayer property*), 146
- `metadata()` (*LinearlyMappedLayer property*), 149
- `metadata()` (*LinuxKernelIntermedSymbols property*), 321
- `metadata()` (*MacKernelIntermedSymbols property*), 344
- `metadata()` (*PdbMSFStream property*), 152
- `metadata()` (*PdbMultiStreamFormat property*), 154
- `metadata()` (*RegistryHive property*), 162
- `metadata()` (*SegmentedLayer property*), 165
- `metadata()` (*TranslationLayerInterface property*), 108
- `metadata()` (*Version1Format property*), 406
- `metadata()` (*Version2Format property*), 408
- `metadata()` (*Version3Format property*), 411
- `metadata()` (*Version4Format property*), 413
- `metadata()` (*Version5Format property*), 416
- `metadata()` (*Version6Format property*), 418
- `metadata()` (*Version7Format property*), 421
- `metadata()` (*VmwareLayer property*), 167
- `metadata()` (*WindowsCrashDump32Layer property*), 127
- `metadata()` (*WindowsIntel property*), 137
- `metadata()` (*WindowsIntel32e property*), 139
- `metadata()` (*WindowsIntelPAE property*), 141
- `metadata()` (*WindowsKernelIntermedSymbols property*), 359
- `metadata()` (*WindowsMixin property*), 144
- `MetadataInterface` (class in *volatility.framework.interfaces.symbols*), 119
- `method_fixed_mapping()` (*KernelPDBScanner method*), 47
- `method_kdbg_offset()` (*KernelPDBScanner method*), 47
- `method_module_offset()` (*KernelPDBScanner method*), 47
- `methods` (*KernelPDBScanner attribute*), 47
- `minimum_address` (*Intel attribute*), 130
- `minimum_address` (*Intel32e attribute*), 132
- `minimum_address` (*IntelPAE attribute*), 135
- `minimum_address` (*WindowsIntel attribute*), 137
- `minimum_address` (*WindowsIntel32e attribute*), 139
- `minimum_address` (*WindowsIntelPAE attribute*), 142
- `minimum_address` (*WindowsMixin attribute*), 144
- `minimum_address()` (*BufferDataLayer property*), 157
- `minimum_address()` (*DataLayerInterface property*), 104
- `minimum_address()` (*FileLayer property*), 159
- `minimum_address()` (*LimeLayer property*), 146
- `minimum_address()` (*LinearlyMappedLayer property*), 149
- `minimum_address()` (*PdbMSFStream property*), 152
- `minimum_address()` (*PdbMultiStreamFormat property*), 154
- `minimum_address()` (*RegistryHive property*), 162
- `minimum_address()` (*SegmentedLayer property*), 165
- `minimum_address()` (*TranslationLayerInterface property*), 108
- `minimum_address()` (*VmwareLayer property*), 167
- `minimum_address()` (*WindowsCrashDump32Layer property*), 127
- `minor()` (*super\_block property*), 334
- `MINORBITS` (*super\_block attribute*), 333
- `mm_struct` (class in *volatility.framework.symbols.linux.extensions*), 326
- `mm_struct.VolTemplateProxy` (class in *volatility.framework.symbols.linux.extensions*), 327
- `MMVAD` (class in *volatility.framework.symbols.windows.extensions*), 373
- `MMVAD.VolTemplateProxy` (class in *volatility.framework.symbols.windows.extensions*), 373
- `MMVAD_SHORT` (class in *volatility.framework.symbols.windows.extensions*), 375
- `MMVAD_SHORT.VolTemplateProxy` (class in *volatility.framework.symbols.windows.extensions*), 375
- `ModDump` (class in *volatility.plugins.windows.moddump*), 271
- `MODIFIED` (*TimeLinerType attribute*), 306
- `ModScan` (class in *volatility.plugins.windows.modscan*), 273
- `Module` (class in *volatility.framework.contexts*), 82
- `module` (class in *volatility.framework.symbols.linux.extensions*), 328
- `module()` (*Context method*), 82
- `module()` (*ContextInterface method*), 100
- `module.VolTemplateProxy` (class in *volatility.framework.symbols.linux.extensions*), 328
- `ModuleCollection` (class in *volatility.framework.contexts*), 84
- `ModuleInterface` (class in *volatility.framework.interfaces.context*), 101
- `Modules` (class in *volatility.plugins.windows.modules*), 275
- `modules()` (*ModuleCollection property*), 84
- `mount` (class in *volatility*

`ity.framework.symbols.linux.extensions)`,  
329  
`mount.VolTemplateProxy` (class in `volatility.framework.symbols.linux.extensions`), 329  
`Multiprocessing` (*Parallelism* attribute), 80  
`MultiRegex` (class in `volatility.framework.layers.scanners.multiregexp`),  
125  
`MultiRequirement` (class in `volatility.framework.configuration.requirements`),  
70  
`MultiStringScanner` (class in `volatility.framework.layers.scanners`), 125  
`MutantScan` (class in `volatility.plugins.windows.mutantscan`), 276  
`MuteProgress` (class in `volatility.cli`), 22

## N

`name` (*CLIRenderer* attribute), 33  
`name` (*CSVRenderer* attribute), 33  
`name` (*PrettyTextRenderer* attribute), 33  
`name` (*QuickTextRenderer* attribute), 33  
`name` () (*BooleanRequirement* property), 60  
`name` () (*BufferDataLayer* property), 157  
`name` () (*BytesRequirement* property), 62  
`name` () (*ChoiceRequirement* property), 63  
`name` () (*ClassRequirement* property), 90  
`name` () (*CMHIVE* property), 387  
`name` () (*Column* property), 115  
`name` () (*ComplexListRequirement* property), 65  
`name` () (*ConfigurableRequirementInterface* property),  
92  
`name` () (*ConstructableRequirementInterface* property),  
94  
`name` () (*DataLayerInterface* property), 104  
`name` () (*FileLayer* property), 159  
`name` () (*Intel* property), 130  
`name` () (*Intel32e* property), 132  
`name` () (*IntelPAE* property), 135  
`name` () (*IntRequirement* property), 66  
`name` () (*LayerListRequirement* property), 68  
`name` () (*LimeLayer* property), 146  
`name` () (*LinearlyMappedLayer* property), 149  
`name` () (*ListRequirement* property), 69  
`name` () (*Module* property), 83  
`name` () (*ModuleInterface* property), 102  
`name` () (*MultiRequirement* property), 71  
`name` () (*PdbMSFStream* property), 152  
`name` () (*PdbMultiStreamFormat* property), 154  
`name` () (*PluginRequirement* property), 72  
`name` () (*RegistryHive* property), 162  
`name` () (*RequirementInterface* property), 97  
`name` () (*SegmentedLayer* property), 165  
`name` () (*SimpleTypeRequirement* property), 98

`name` () (*SizedModule* property), 85  
`name` () (*StringRequirement* property), 74  
`name` () (*SymbolInterface* property), 121  
`name` () (*SymbolTableRequirement* property), 76  
`name` () (*TranslationLayerInterface* property), 108  
`name` () (*TranslationLayerRequirement* property), 77  
`name` () (*URIRequirement* property), 79  
`name` () (*VmwareLayer* property), 167  
`name` () (*WindowsCrashDump32Layer* property), 127  
`name` () (*WindowsIntel* property), 137  
`name` () (*WindowsIntel32e* property), 139  
`name` () (*WindowsIntelPAE* property), 142  
`name` () (*WindowsMixin* property), 144  
`name_as_str` () (*qstr* method), 331  
`name_strip` () (*PdbReader* method), 398  
`NameInfo` () (*OBJECT\_HEADER* property), 376  
`natives` () (*BaseSymbolTableInterface* property), 119  
`natives` () (*BashIntermedSymbols* property), 341  
`natives` () (*IntermediateSymbolTable* property), 403  
`natives` () (*ISFormatTable* property), 400  
`natives` () (*LinuxKernelIntermedSymbols* property),  
321  
`natives` () (*MacKernelIntermedSymbols* property),  
344  
`natives` () (*NativeTable* property), 423  
`natives` () (*NativeTableInterface* property), 120  
`natives` () (*SymbolTableInterface* property), 124  
`natives` () (*Version1Format* property), 406  
`natives` () (*Version2Format* property), 408  
`natives` () (*Version3Format* property), 411  
`natives` () (*Version4Format* property), 413  
`natives` () (*Version5Format* property), 416  
`natives` () (*Version6Format* property), 418  
`natives` () (*Version7Format* property), 421  
`natives` () (*WindowsKernelIntermedSymbols* property), 359  
`NativeTable` (class in `volatility.framework.symbols.native`), 422  
`NativeTableInterface` (class in `volatility.framework.interfaces.symbols`), 119  
`Netstat` (class in `volatility.plugins.mac.netstat`), 233  
`new_requirement` () (*ComplexListRequirement* method), 65  
`new_requirement` () (*LayerListRequirement* method), 68  
`noninheritable` (class in `volatility.framework`), 34  
`NONPAGED` (*PoolType* attribute), 281  
`NotApplicableValue` (class in `volatility.framework.renderers`), 307  
`NotAvailableValue` (class in `volatility.framework.renderers`), 307  
`NullFileConsumer` (class in `volatility.cli.volshell.generic`), 23  
`numerator` (*Bin* attribute), 311

numerator (*BitField attribute*), 172  
 numerator (*Boolean attribute*), 174  
 numerator (*Char attribute*), 180  
 numerator (*Enumeration attribute*), 184  
 numerator (*Hex attribute*), 312  
 numerator (*Integer attribute*), 189  
 numerator (*Pointer attribute*), 191

## O

object () (*Context method*), 82  
 object () (*ContextInterface method*), 100  
 object () (*Module method*), 83  
 object () (*ModuleInterface method*), 102  
 object () (*SizedModule method*), 85  
 object\_from\_symbol () (*Module method*), 83  
 object\_from\_symbol () (*ModuleInterface method*), 102  
 object\_from\_symbol () (*SizedModule method*), 86  
 OBJECT\_HEADER (class in *volatility.framework.symbols.windows.extensions*), 376  
 object\_header () (*DEVICE\_OBJECT method*), 361  
 object\_header () (*DRIVER\_OBJECT method*), 362  
 object\_header () (*EPROCESS method*), 364  
 object\_header () (*ExecutiveObject method*), 368  
 object\_header () (*FILE\_OBJECT method*), 369  
 object\_header () (*KMUTANT method*), 370  
 object\_header () (*OBJECT\_SYMBOLIC\_LINK method*), 379  
 OBJECT\_HEADER.VolTemplateProxy (class in *volatility.framework.symbols.windows.extensions*), 377  
 OBJECT\_SYMBOLIC\_LINK (class in *volatility.framework.symbols.windows.extensions*), 378  
 OBJECT\_SYMBOLIC\_LINK.VolTemplateProxy (class in *volatility.framework.symbols.windows.extensions*), 378  
 ObjectInformation (class in *volatility.framework.interfaces.objects*), 109  
 ObjectInterface (class in *volatility.framework.interfaces.objects*), 110  
 ObjectInterface.VolTemplateProxy (class in *volatility.framework.interfaces.objects*), 110  
 ObjectTemplate (class in *volatility.framework.objects.templates*), 202  
 Off (*Parallelism attribute*), 80  
 offset () (*Module property*), 84  
 offset () (*ModuleInterface property*), 102  
 offset () (*SizedModule property*), 86  
 omap\_lookup () (*PdbReader method*), 398  
 open () (*ResourceAccessor method*), 163

optional () (*BooleanRequirement property*), 60  
 optional () (*BytesRequirement property*), 62  
 optional () (*ChoiceRequirement property*), 63  
 optional () (*ClassRequirement property*), 90  
 optional () (*ComplexListRequirement property*), 65  
 optional () (*ConfigurableRequirementInterface property*), 92  
 optional () (*ConstructableRequirementInterface property*), 94  
 optional () (*in module volatility.cli.text\_renderer*), 34  
 optional () (*IntRequirement property*), 66  
 optional () (*LayerListRequirement property*), 68  
 optional () (*ListRequirement property*), 70  
 optional () (*MultiRequirement property*), 71  
 optional () (*PluginRequirement property*), 73  
 optional () (*RequirementInterface property*), 97  
 optional () (*SimpleTypeRequirement property*), 98  
 optional () (*StringRequirement property*), 74  
 optional () (*SymbolTableRequirement property*), 76  
 optional () (*TranslationLayerRequirement property*), 77  
 optional () (*URIRequirement property*), 79  
 os (*LinuxBannerCache attribute*), 38  
 os (*MacBannerCache attribute*), 42  
 os (*SymbolBannerCache attribute*), 52  
 os\_distinguisher () (*in module volatility.plugins.windows.poolscanner*), 282  
 overlap (*PageMapScanner attribute*), 56  
 overlap (*PdbSignatureScanner attribute*), 48  
 owning\_process () (*ETHREAD method*), 365

## P

PACKAGE\_VERSION (in *module volatility.framework.constants*), 80  
 PAGE\_SHIFT (in *module volatility.framework.constants.linux*), 80  
 page\_size (*Intel attribute*), 130  
 page\_size (*Intel32e attribute*), 133  
 page\_size (*IntelPAE attribute*), 135  
 page\_size (*WindowsIntel attribute*), 137  
 page\_size (*WindowsIntel32e attribute*), 139  
 page\_size (*WindowsIntelPAE attribute*), 142  
 page\_size (*WindowsMixin attribute*), 144  
 page\_size () (*PdbMultiStreamFormat property*), 154  
 PAGED (*PoolType attribute*), 281  
 PagedInvalidAddressException, 424  
 PageMapScanner (class in *volatility.framework.automagic.windows*), 56  
 Parallelism (class in *volatility.framework.constants*), 80  
 PARALLELISM (in *module volatility.framework.constants*), 80  
 parent () (*TreeNode property*), 117, 309

`parent_path()` (in module `volatility.framework.interfaces.configuration`), 99

`parse_data()` (*Certificates method*), 246

`parse_string()` (*PdbReader static method*), 398

`parse_userassist_data()` (*UserAssist method*), 253

`partition()` (*Bytes method*), 177

`partition()` (*HexBytes method*), 314

`partition()` (*String method*), 196

`path()` (*dentry method*), 323

`path()` (*TreeNode property*), 117, 309

`path_changed()` (*TreeNode method*), 117, 309

`path_depth()` (in module `volatility.framework.interfaces.configuration`), 99

`path_depth()` (*TreeGrid static method*), 116, 308

`path_depth()` (*TreeNode property*), 117, 309

`path_for_file()` (*LinuxUtilities class method*), 41

`path_join()` (in module `volatility.framework.interfaces.configuration`), 99

`path_sep` (*TreeGrid attribute*), 308

`pdb_age()` (*WindowsMetadata property*), 422

`pdb_guid()` (*WindowsMetadata property*), 422

`pdb_layer_name()` (*PdbReader property*), 398

`pdb_symbol_table()` (*PdbMSFStream property*), 152

`pdb_symbol_table()` (*PdbMultiStreamFormat property*), 154

`PdbMSFStream` (class in `volatility.framework.layers.msf`), 150

`PdbMultiStreamFormat` (class in `volatility.framework.layers.msf`), 153

`PdbReader` (class in `volatility.framework.symbols.windows.pdbconv`), 397

`PdbRetriever` (class in `volatility.framework.symbols.windows.pdbconv`), 398

`PdbSignatureScanner` (class in `volatility.framework.automagic.pdbscan`), 48

`pe_version()` (*WindowsMetadata property*), 422

`pe_version_string()` (*WindowsMetadata property*), 422

`perm_flags` (*vm\_area\_struct attribute*), 338

`PHYSICAL_DEFAULT` (*PsList attribute*), 284

`PHYSICAL_DEFAULT` (*PsTree attribute*), 288

`PluginInterface` (class in `volatility.framework.interfaces.plugins`), 113

`PluginRequirement` (class in `volatility.framework.configuration.requirements`), 72

`PluginRequirementException`, 424

`PLUGINS_PATH` (in module `volatility.framework.constants`), 80

`PluginVersionException`, 424

`Pointer` (class in `volatility.framework.objects`), 189

`Pointer.VolTemplateProxy` (class in `volatility.framework.objects`), 190

`pointer_to_string()` (in module `volatility.framework.objects.utility`), 204

`POOL_HEADER` (class in `volatility.framework.symbols.windows.extensions`), 379

`POOL_HEADER.VolTemplateProxy` (class in `volatility.framework.symbols.windows.extensions`), 379

`pool_scan()` (*PoolScanner class method*), 280

`PoolConstraint` (class in `volatility.plugins.windows.poolscanner`), 278

`PoolHeaderScanner` (class in `volatility.plugins.windows.poolscanner`), 278

`PoolScanner` (class in `volatility.plugins.windows.poolscanner`), 278

`PoolType` (class in `volatility.plugins.windows.poolscanner`), 281

`populate()` (*TreeGrid method*), 116, 308

`populate_config()` (*CommandLine method*), 22

`populate_config()` (*VolShell method*), 23

`populate_requirements_argparse()` (*CommandLine method*), 22

`populate_requirements_argparse()` (*VolShell method*), 23

`populated()` (*TreeGrid property*), 116, 308

`possible_architectures` (*Disassembly attribute*), 115

`preprocess()` (*MultiRegex method*), 126

`PrettyTextRenderer` (class in `volatility.cli.text_renderer`), 33

`PrimitiveObject` (class in `volatility.framework.objects`), 192

`PrimitiveObject.VolTemplateProxy` (class in `volatility.framework.objects`), 192

`PrintedProgress` (class in `volatility.cli`), 22

`PrintKey` (class in `volatility.plugins.windows.registry.printkey`), 251

`priority` (*AutomagicInterface attribute*), 88

`priority` (*BufferDataLayer attribute*), 157

`priority` (*ConstructionMagic attribute*), 37

`priority` (*FileLayer attribute*), 159

`priority` (*Intel attribute*), 130

`priority` (*Intel32e attribute*), 133

`priority` (*IntelPAE attribute*), 135

`priority` (*KernelPDBScanner attribute*), 47

`priority` (*LayerStacker attribute*), 50

`priority` (*LimeLayer attribute*), 146

`priority` (*LinuxBannerCache attribute*), 38

`priority` (*LinuxSymbolFinder attribute*), 40

`priority` (*MacBannerCache attribute*), 42



- priority (*MacSymbolFinder* attribute), 44
  - priority (*SymbolBannerCache* attribute), 52
  - priority (*SymbolFinder* attribute), 53
  - priority (*VmwareLayer* attribute), 167
  - priority (*WindowsCrashDump32Layer* attribute), 128
  - priority (*WindowsIntel* attribute), 137
  - priority (*WindowsIntel32e* attribute), 139
  - priority (*WindowsIntelPAE* attribute), 142
  - priority (*WindowsMixin* attribute), 144
  - priority (*WinSwapLayers* attribute), 57
  - priority (*WintelHelper* attribute), 59
  - proc (class in *volatility.framework.symbols.mac.extensions*), 348
  - proc.VolTemplateProxy (class in *volatility.framework.symbols.mac.extensions*), 348
  - ProcDump (class in *volatility.plugins.windows.procdump*), 282
  - process\_exceptions() (*CommandLine* method), 22
  - process\_exceptions() (*VolShell* method), 23
  - process\_types() (*PdbReader* method), 398
  - produce\_file() (*Bash* method), 205, 222
  - produce\_file() (*Certificates* method), 246
  - produce\_file() (*Check\_afinfo* method), 207
  - produce\_file() (*Check\_syscall* method), 208, 223, 244
  - produce\_file() (*Check\_sysctl* method), 225
  - produce\_file() (*Check\_trap\_table* method), 226
  - produce\_file() (*CmdLine* method), 256
  - produce\_file() (*ConfigWriter* method), 303
  - produce\_file() (*DllDump* method), 258
  - produce\_file() (*DllList* method), 259
  - produce\_file() (*DriverIrp* method), 261
  - produce\_file() (*DriverScan* method), 262
  - produce\_file() (*Elfs* method), 210
  - produce\_file() (*FileScan* method), 264
  - produce\_file() (*Handles* method), 266
  - produce\_file() (*HiveList* method), 248
  - produce\_file() (*HiveScan* method), 250
  - produce\_file() (*Ifconfig* method), 228
  - produce\_file() (*Info* method), 268
  - produce\_file() (*LayerWriter* method), 305
  - produce\_file() (*Lsmmod* method), 212, 230
  - produce\_file() (*Lsof* method), 213
  - produce\_file() (*Lsof* method), 231
  - produce\_file() (*Malfind* method), 215, 233, 270
  - produce\_file() (*Maps* method), 216, 236
  - produce\_file() (*ModDump* method), 272
  - produce\_file() (*ModScan* method), 274
  - produce\_file() (*Modules* method), 276
  - produce\_file() (*MutantScan* method), 277
  - produce\_file() (*Netstat* method), 234
  - produce\_file() (*PluginInterface* method), 114
  - produce\_file() (*PoolScanner* method), 281
  - produce\_file() (*PrintKey* method), 252
  - produce\_file() (*ProcDump* method), 283
  - produce\_file() (*Psaux* method), 237
  - produce\_file() (*PsList* method), 218, 239, 285
  - produce\_file() (*PsScan* method), 287
  - produce\_file() (*PsTree* method), 220, 241, 289
  - produce\_file() (*SSDT* method), 291
  - produce\_file() (*Statistics* method), 255
  - produce\_file() (*Strings* method), 293
  - produce\_file() (*SymlinkScan* method), 295
  - produce\_file() (*Tasks* method), 242
  - produce\_file() (*Timeliner* method), 306
  - produce\_file() (*UserAssist* method), 253
  - produce\_file() (*VadDump* method), 296
  - produce\_file() (*VadInfo* method), 298
  - produce\_file() (*VerInfo* method), 300
  - produce\_file() (*VirtMap* method), 301
  - produce\_file() (*Volshell* method), 25, 27, 30, 32
  - ProgressCallback (in module *volatility.framework.constants*), 80
  - protect\_values() (*VadInfo* class method), 298
  - provides (*LinuxKernelIntermedSymbols* attribute), 321
  - provides (*MacKernelIntermedSymbols* attribute), 344
  - provides (*WindowsCrashDump32Layer* attribute), 128
  - Psaux (class in *volatility.plugins.mac.psaux*), 236
  - PsList (class in *volatility.plugins.linux.pslist*), 217
  - PsList (class in *volatility.plugins.mac.pslist*), 238
  - PsList (class in *volatility.plugins.windows.pslist*), 284
  - PsScan (class in *volatility.plugins.windows.psscan*), 286
  - PsTree (class in *volatility.plugins.linux.pstree*), 219
  - PsTree (class in *volatility.plugins.mac.pstree*), 240
  - PsTree (class in *volatility.plugins.windows.pstree*), 288
  - PYTHONPATH, 19
- ## Q
- qstr (class in *volatility.framework.symbols.linux.extensions*), 330
  - qstr.VolTemplateProxy (class in *volatility.framework.symbols.linux.extensions*), 330
  - queue\_entry (class in *volatility.framework.symbols.mac.extensions*), 349
  - queue\_entry.VolTemplateProxy (class in *volatility.framework.symbols.mac.extensions*), 349
  - QuickTextRenderer (class in *volatility.cli.text\_renderer*), 33
  - quoted\_optional() (in module *volatility.cli.text\_renderer*), 34

## R

- `read` (*Intel* attribute), 130
- `read` (*Intel32e* attribute), 133
- `read` (*IntelPAE* attribute), 135
- `read` (*LimeLayer* attribute), 147
- `read` (*LinearlyMappedLayer* attribute), 149
- `read` (*PdbMSFStream* attribute), 152
- `read` (*PdbMultiStreamFormat* attribute), 154
- `read` (*RegistryHive* attribute), 162
- `read` (*SegmentedLayer* attribute), 165
- `read` (*TranslationLayerInterface* attribute), 108
- `read` (*VmwareLayer* attribute), 167
- `read` (*WindowsCrashDump32Layer* attribute), 128
- `read` (*WindowsIntel* attribute), 137
- `read` (*WindowsIntel32e* attribute), 139
- `read` (*WindowsIntelPAE* attribute), 142
- `read` (*WindowsMixin* attribute), 144
- `read()` (*BufferDataLayer* method), 157
- `read()` (*DataLayerInterface* method), 104
- `read()` (*FileLayer* method), 159
- `read()` (*LayerContainer* method), 106
- `read_dbi_stream()` (*PdbReader* method), 398
- `read_necessary_streams()` (*PdbReader* method), 398
- `read_pdb_info_stream()` (*PdbReader* method), 398
- `read_streams()` (*PdbMultiStreamFormat* method), 154
- `read_symbol_stream()` (*PdbReader* method), 398
- `read_tpi_stream()` (*PdbReader* method), 398
- `ReadOnlyMapping` (class in *volatility.framework.interfaces.objects*), 111
- `real` (*Bin* attribute), 311
- `real` (*BitField* attribute), 172
- `real` (*Boolean* attribute), 174
- `real` (*Char* attribute), 180
- `real` (*Enumeration* attribute), 184
- `real` (*Float* attribute), 186
- `real` (*Hex* attribute), 312
- `real` (*Integer* attribute), 189
- `real` (*Pointer* attribute), 191
- `reconstruct()` (*IMAGE\_DOS\_HEADER* method), 384
- `record_cached_validations()` (in module *volatility.schemas*), 425
- `recurse_symbol_fulfiller()` (*KernelPDB-Scanner* method), 47
- `ReferenceTemplate` (class in *volatility.framework.objects.templates*), 203
- `REG_BINARY` (*RegValueTypes* attribute), 393
- `REG_DWORD` (*RegValueTypes* attribute), 393
- `REG_DWORD_BIG_ENDIAN` (*RegValueTypes* attribute), 393
- `REG_EXPAND_SZ` (*RegValueTypes* attribute), 393
- `REG_FULL_RESOURCE_DESCRIPTOR` (*RegValueTypes* attribute), 393
- `REG_LINK` (*RegValueTypes* attribute), 393
- `REG_MULTI_SZ` (*RegValueTypes* attribute), 393
- `REG_NONE` (*RegValueTypes* attribute), 393
- `REG_QWORD` (*RegValueTypes* attribute), 393
- `REG_RESOURCE_LIST` (*RegValueTypes* attribute), 393
- `REG_RESOURCE_REQUIREMENTS_LIST` (*RegValueTypes* attribute), 393
- `REG_SZ` (*RegValueTypes* attribute), 393
- `REG_UNKNOWN` (*RegValueTypes* attribute), 393
- `RegExScanner` (class in *volatility.framework.layers.scanners*), 125
- `RegistryFormatException`, 160
- `RegistryHive` (class in *volatility.framework.layers.registry*), 160
- `RegistryInvalidIndex`, 163
- `RegKeyFlags` (class in *volatility.framework.symbols.windows.extensions.registry*), 393
- `RegValueTypes` (class in *volatility.framework.symbols.windows.extensions.registry*), 393
- `relative_child_offset()` (*AggregateType.VolTemplateProxy* class method), 169
- `relative_child_offset()` (*Array.VolTemplateProxy* class method), 170
- `relative_child_offset()` (*BitField.VolTemplateProxy* class method), 171
- `relative_child_offset()` (*Boolean.VolTemplateProxy* class method), 173
- `relative_child_offset()` (*Bytes.VolTemplateProxy* class method), 175
- `relative_child_offset()` (*Char.VolTemplateProxy* class method), 179
- `relative_child_offset()` (*ClassType.VolTemplateProxy* class method), 181
- `relative_child_offset()` (*CM\_KEY\_BODY.VolTemplateProxy* class method), 388
- `relative_child_offset()` (*CM\_KEY\_NODE.VolTemplateProxy* class method), 389
- `relative_child_offset()` (*CM\_KEY\_VALUE.VolTemplateProxy* class method), 391
- `relative_child_offset()` (*CMHIVE.VolTemplateProxy* class method), 387
- `relative_child_offset()` (*den-*

*try.VolTemplateProxy class method*), 322

*relative\_child\_offset()* (*DEVICE\_OBJECT.VolTemplateProxy class method*), 360

*relative\_child\_offset()* (*DRIVER\_OBJECT.VolTemplateProxy class method*), 362

*relative\_child\_offset()* (*Enumeration.VolTemplateProxy class method*), 183

*relative\_child\_offset()* (*EPROCESS.VolTemplateProxy class method*), 363

*relative\_child\_offset()* (*ETHREAD.VolTemplateProxy class method*), 364

*relative\_child\_offset()* (*EX\_FAST\_REF.VolTemplateProxy class method*), 366

*relative\_child\_offset()* (*ExecutiveObject.VolTemplateProxy class method*), 367

*relative\_child\_offset()* (*FILE\_OBJECT.VolTemplateProxy class method*), 368

*relative\_child\_offset()* (*fileglob.VolTemplateProxy class method*), 345

*relative\_child\_offset()* (*files\_struct.VolTemplateProxy class method*), 323

*relative\_child\_offset()* (*Float.VolTemplateProxy class method*), 185

*relative\_child\_offset()* (*fs\_struct.VolTemplateProxy class method*), 325

*relative\_child\_offset()* (*Function.VolTemplateProxy class method*), 187

*relative\_child\_offset()* (*GenericIntelProcess.VolTemplateProxy class method*), 318

*relative\_child\_offset()* (*hist\_entry.VolTemplateProxy class method*), 338

*relative\_child\_offset()* (*HMAP\_ENTRY.VolTemplateProxy class method*), 392

*relative\_child\_offset()* (*ifnet.VolTemplateProxy class method*), 346

*relative\_child\_offset()* (*IMAGE\_DOS\_HEADER.VolTemplateProxy class method*), 383

*relative\_child\_offset()* (*IMAGE\_NT\_HEADERS.VolTemplateProxy class method*), 385

*relative\_child\_offset()* (*inpcb.VolTemplateProxy class method*), 347

*relative\_child\_offset()* (*Integer.VolTemplateProxy class method*), 188

*relative\_child\_offset()* (*KDDEBUGGER\_DATA64.VolTemplateProxy class method*), 382

*relative\_child\_offset()* (*KMUTANT.VolTemplateProxy class method*), 370

*relative\_child\_offset()* (*KSYSTEM\_TIME.VolTemplateProxy class method*), 371

*relative\_child\_offset()* (*LIST\_ENTRY.VolTemplateProxy class method*), 372

*relative\_child\_offset()* (*list\_head.VolTemplateProxy class method*), 326

*relative\_child\_offset()* (*mm\_struct.VolTemplateProxy class method*), 327

*relative\_child\_offset()* (*MMVAD.VolTemplateProxy class method*), 373

*relative\_child\_offset()* (*MMVAD\_SHORT.VolTemplateProxy class method*), 375

*relative\_child\_offset()* (*module.VolTemplateProxy class method*), 328

*relative\_child\_offset()* (*mount.VolTemplateProxy class method*), 329

*relative\_child\_offset()* (*OBJECT\_HEADER.VolTemplateProxy class method*), 377

*relative\_child\_offset()* (*OBJECT\_SYMBOLIC\_LINK.VolTemplateProxy class method*), 378

*relative\_child\_offset()* (*ObjectInterface.VolTemplateProxy class method*), 110

*relative\_child\_offset()* (*ObjectTemplate method*), 202

*relative\_child\_offset()* (*Pointer.VolTemplateProxy class method*), 190

*relative\_child\_offset()* (*POOL\_HEADER.VolTemplateProxy class method*), 379

*relative\_child\_offset()* (*PrimitiveObject.VolTemplateProxy class method*), 192

*relative\_child\_offset()* (*proc.VolTemplateProxy class method*), 348

*relative\_child\_offset()* (*qstr.VolTemplateProxy class method*), 330

*relative\_child\_offset()* (*queue\_entry.VolTemplateProxy class method*),

349  
relative\_child\_offset() (*ReferenceTemplate*  
method), 203  
relative\_child\_offset() (*SER-*  
*VICE\_HEADER.VolTemplateProxy*  
class  
method), 394  
relative\_child\_offset() (*SER-*  
*VICE\_RECORD.VolTemplateProxy*  
class  
method), 395  
relative\_child\_offset() (*sock-*  
*addr.VolTemplateProxy* class method), 350  
relative\_child\_offset() (*sock-*  
*addr\_dl.VolTemplateProxy* class  
method),  
352  
relative\_child\_offset() (*socket.VolTemplateProxy*  
class  
method),  
353  
relative\_child\_offset() (*String.VolTemplateProxy*  
class  
method),  
193  
relative\_child\_offset() (*struct\_file.VolTemplateProxy*  
class  
method),  
332  
relative\_child\_offset() (*Struct-*  
*Type.VolTemplateProxy* class method), 198  
relative\_child\_offset() (*su-*  
*per\_block.VolTemplateProxy* class  
method),  
333  
relative\_child\_offset() (*Symbol-*  
*Space.UnresolvedTemplate* method), 316  
relative\_child\_offset() (*task\_struct.VolTemplateProxy*  
class  
method),  
334  
relative\_child\_offset() (*Template* method),  
112  
relative\_child\_offset() (*UNI-*  
*CODE\_STRING.VolTemplateProxy*  
class  
method), 381  
relative\_child\_offset() (*Union-*  
*Type.VolTemplateProxy* class method), 199  
relative\_child\_offset() (*vfs-*  
*mount.VolTemplateProxy* class  
method),  
335  
relative\_child\_offset() (*vm\_area\_struct.VolTemplateProxy*  
class  
method), 337  
relative\_child\_offset() (*vm\_map\_entry.VolTemplateProxy*  
class  
method), 354  
relative\_child\_offset() (*vm\_map\_object.VolTemplateProxy*  
class  
method), 355  
relative\_child\_offset() (*vn-*  
*ode.VolTemplateProxy* class method), 356  
relative\_child\_offset() (*Void.VolTemplateProxy* class  
method), 201  
remove() (*SymbolSpace* method), 318  
remove\_requirement() (*BooleanRequirement*  
method), 61  
remove\_requirement() (*BytesRequirement*  
method), 62  
remove\_requirement() (*ChoiceRequirement*  
method), 63  
remove\_requirement() (*ClassRequirement*  
method), 90  
remove\_requirement() (*ComplexListRequirement*  
method), 65  
remove\_requirement() (*ConfigurableRequire-*  
*mentInterface* method), 92  
remove\_requirement() (*ConstructableRequire-*  
*mentInterface* method), 94  
remove\_requirement() (*IntRequirement* method),  
66  
remove\_requirement() (*LayerListRequirement*  
method), 68  
remove\_requirement() (*ListRequirement* method),  
70  
remove\_requirement() (*MultiRequirement*  
method), 71  
remove\_requirement() (*PluginRequirement*  
method), 73  
remove\_requirement() (*RequirementInterface*  
method), 97  
remove\_requirement() (*SimpleTypeRequirement*  
method), 99  
remove\_requirement() (*StringRequirement*  
method), 74  
remove\_requirement() (*SymbolTableRequirement*  
method), 76  
remove\_requirement() (*TranslationLayerRequire-*  
*ment* method), 78  
remove\_requirement() (*URIRequirement*  
method), 79  
render() (*CLIRenderer* method), 33  
render() (*CSVRenderer* method), 33  
render() (*PrettyTextRenderer* method), 33  
render() (*QuickTextRenderer* method), 34  
render() (*Renderer* method), 115  
render\_treegrid() (*Volshell* method), 25, 27, 30,  
32  
Renderer (class in *volatil-*  
*ity.framework.interfaces.renderers*), 115  
replace() (*Bytes* method), 177  
replace() (*HexBytes* method), 314  
replace() (*String* method), 196  
replace\_child() (*AggregateType.VolTemplateProxy*  
class  
method), 169  
replace\_child() (*Array.VolTemplateProxy* class



`method`), 170  
`replace_child()` (*BitField.VolTemplateProxy class method*), 171  
`replace_child()` (*Boolean.VolTemplateProxy class method*), 173  
`replace_child()` (*Bytes.VolTemplateProxy class method*), 175  
`replace_child()` (*Char.VolTemplateProxy class method*), 179  
`replace_child()` (*ClassType.VolTemplateProxy class method*), 181  
`replace_child()` (*CM\_KEY\_BODY.VolTemplateProxy class method*), 388  
`replace_child()` (*CM\_KEY\_NODE.VolTemplateProxy class method*), 389  
`replace_child()` (*CM\_KEY\_VALUE.VolTemplateProxy class method*), 391  
`replace_child()` (*CMHIVE.VolTemplateProxy class method*), 387  
`replace_child()` (*dentry.VolTemplateProxy class method*), 322  
`replace_child()` (*DEVICE\_OBJECT.VolTemplateProxy class method*), 360  
`replace_child()` (*DRIVER\_OBJECT.VolTemplateProxy class method*), 362  
`replace_child()` (*Enumeration.VolTemplateProxy class method*), 183  
`replace_child()` (*EPROCESS.VolTemplateProxy class method*), 363  
`replace_child()` (*ETHREAD.VolTemplateProxy class method*), 364  
`replace_child()` (*EX\_FAST\_REF.VolTemplateProxy class method*), 366  
`replace_child()` (*ExecutiveObject.VolTemplateProxy class method*), 367  
`replace_child()` (*FILE\_OBJECT.VolTemplateProxy class method*), 368  
`replace_child()` (*fileglob.VolTemplateProxy class method*), 345  
`replace_child()` (*files\_struct.VolTemplateProxy class method*), 323  
`replace_child()` (*Float.VolTemplateProxy class method*), 185  
`replace_child()` (*fs\_struct.VolTemplateProxy class method*), 325  
`replace_child()` (*Function.VolTemplateProxy class method*), 187  
`replace_child()` (*GenericIntelProcess.VolTemplateProxy class method*), 318  
`replace_child()` (*hist\_entry.VolTemplateProxy class method*), 338  
`replace_child()` (*HMAP\_ENTRY.VolTemplateProxy class method*), 392  
`replace_child()` (*ifnet.VolTemplateProxy class method*), 346  
`replace_child()` (*IM-AGE\_DOS\_HEADER.VolTemplateProxy class method*), 383  
`replace_child()` (*IM-AGE\_NT\_HEADERS.VolTemplateProxy class method*), 385  
`replace_child()` (*inpcb.VolTemplateProxy class method*), 347  
`replace_child()` (*Integer.VolTemplateProxy class method*), 188  
`replace_child()` (*KDDEBUG-GER\_DATA64.VolTemplateProxy class method*), 382  
`replace_child()` (*KMUTANT.VolTemplateProxy class method*), 370  
`replace_child()` (*KSYS-TEM\_TIME.VolTemplateProxy class method*), 371  
`replace_child()` (*LIST\_ENTRY.VolTemplateProxy class method*), 372  
`replace_child()` (*list\_head.VolTemplateProxy class method*), 326  
`replace_child()` (*mm\_struct.VolTemplateProxy class method*), 327  
`replace_child()` (*MMVAD.VolTemplateProxy class method*), 373  
`replace_child()` (*MM-VAD\_SHORT.VolTemplateProxy class method*), 375  
`replace_child()` (*module.VolTemplateProxy class method*), 328  
`replace_child()` (*mount.VolTemplateProxy class method*), 329  
`replace_child()` (*OBJECT\_HEADER.VolTemplateProxy class method*), 377  
`replace_child()` (*OBJECT\_SYMBOLIC\_LINK.VolTemplateProxy class method*), 378  
`replace_child()` (*ObjectInterface.VolTemplateProxy class method*), 110  
`replace_child()` (*ObjectTemplate method*), 202  
`replace_child()` (*Pointer.VolTemplateProxy class method*), 190  
`replace_child()` (*POOL\_HEADER.VolTemplateProxy class method*), 379  
`replace_child()` (*PrimitiveObject.VolTemplateProxy class method*), 192  
`replace_child()` (*proc.VolTemplateProxy class method*), 348  
`replace_child()` (*qstr.VolTemplateProxy class method*), 331

`replace_child()` (*queue\_entry.VolTemplateProxy class method*), 349

`replace_child()` (*ReferenceTemplate method*), 203

`replace_child()` (*SER-VICE\_HEADER.VolTemplateProxy class method*), 394

`replace_child()` (*SER-VICE\_RECORD.VolTemplateProxy class method*), 395

`replace_child()` (*sockaddr.VolTemplateProxy class method*), 351

`replace_child()` (*sockaddr\_dl.VolTemplateProxy class method*), 352

`replace_child()` (*socket.VolTemplateProxy class method*), 353

`replace_child()` (*String.VolTemplateProxy class method*), 193

`replace_child()` (*struct\_file.VolTemplateProxy class method*), 332

`replace_child()` (*StructType.VolTemplateProxy class method*), 198

`replace_child()` (*super\_block.VolTemplateProxy class method*), 333

`replace_child()` (*Symbol-Space.UnresolvedTemplate method*), 316

`replace_child()` (*task\_struct.VolTemplateProxy class method*), 334

`replace_child()` (*Template method*), 112

`replace_child()` (*UNICODE\_STRING.VolTemplateProxy class method*), 381

`replace_child()` (*UnionType.VolTemplateProxy class method*), 199

`replace_child()` (*vfsmount.VolTemplateProxy class method*), 335

`replace_child()` (*vm\_area\_struct.VolTemplateProxy class method*), 337

`replace_child()` (*vm\_map\_entry.VolTemplateProxy class method*), 354

`replace_child()` (*vm\_map\_object.VolTemplateProxy class method*), 355

`replace_child()` (*vnnode.VolTemplateProxy class method*), 357

`replace_child()` (*Void.VolTemplateProxy class method*), 201

`replace_forward_references()` (*PdbReader method*), 398

`replace_header_field()` (*IMAGE\_DOS\_HEADER method*), 384

`require_interface_version()` (*in module volatility.framework*), 34

*RequirementInterface* (*class in volatility.framework.interfaces.configuration*), 96

`requirements()` (*BooleanRequirement property*), 61

`requirements()` (*BytesRequirement property*), 62

`requirements()` (*ChoiceRequirement property*), 63

`requirements()` (*ClassRequirement property*), 90

`requirements()` (*ComplexListRequirement property*), 65

`requirements()` (*ConfigurableRequirementInterface property*), 92

`requirements()` (*ConstructableRequirementInterface property*), 94

`requirements()` (*IntRequirement property*), 66

`requirements()` (*LayerListRequirement property*), 68

`requirements()` (*ListRequirement property*), 70

`requirements()` (*MultiRequirement property*), 71

`requirements()` (*PluginRequirement property*), 73

`requirements()` (*RequirementInterface property*), 97

`requirements()` (*SimpleTypeRequirement property*), 99

`requirements()` (*StringRequirement property*), 74

`requirements()` (*SymbolTableRequirement property*), 76

`requirements()` (*TranslationLayerRequirement property*), 78

`requirements()` (*URIRequirement property*), 79

`reset()` (*PdbReader method*), 398

*ResourceAccessor* (*class in volatility.framework.layers.resources*), 163

`retrieve_pdb()` (*PdbRetriever method*), 398

`rfind()` (*Bytes method*), 177

`rfind()` (*HexBytes method*), 314

`rfind()` (*String method*), 196

`rindex()` (*Bytes method*), 177

`rindex()` (*HexBytes method*), 314

`rindex()` (*String method*), 197

`rjust()` (*Bytes method*), 178

`rjust()` (*HexBytes method*), 314

`rjust()` (*String method*), 197

`root_cell_offset()` (*RegistryHive property*), 162

`round()` (*in module volatility.framework.renderers.conversion*), 310

`row_count()` (*TreeGrid property*), 308

`rpartition()` (*Bytes method*), 178

`rpartition()` (*HexBytes method*), 315

`rpartition()` (*String method*), 197

`rsplit()` (*Bytes method*), 178

`rsplit()` (*HexBytes method*), 315

`rsplit()` (*String method*), 197

`rstrip()` (*Bytes method*), 178

`rstrip()` (*HexBytes method*), 315

`rstrip()` (*String method*), 197

`run()` (*Bash method*), 205, 222

`run()` (*Certificates method*), 246

`run()` (*Check\_afinfo method*), 207

`run()` (*Check\_syscall method*), 208, 224, 244

run() (*Check\_sysctl method*), 225  
 run() (*Check\_trap\_table method*), 227  
 run() (*CmdLine method*), 257  
 run() (*CommandLine method*), 22  
 run() (*ConfigWriter method*), 303  
 run() (*DllDump method*), 258  
 run() (*DllList method*), 260  
 run() (*DriverIrp method*), 261  
 run() (*DriverScan method*), 263  
 run() (*Elfs method*), 210  
 run() (*FileScan method*), 264  
 run() (*Handles method*), 267  
 run() (*HiveList method*), 248  
 run() (*HiveScan method*), 250  
 run() (*Ifconfig method*), 228  
 run() (*in module volatility.framework.automagic*), 35  
 run() (*Info method*), 268  
 run() (*LayerWriter method*), 305  
 run() (*Lsmode method*), 212, 230  
 run() (*Lsof method*), 213  
 run() (*lsof method*), 231  
 run() (*Malfind method*), 215, 233, 270  
 run() (*Maps method*), 216, 236  
 run() (*ModDump method*), 272  
 run() (*ModScan method*), 274  
 run() (*Modules method*), 276  
 run() (*MutantScan method*), 277  
 run() (*Netstat method*), 234  
 run() (*PluginInterface method*), 114  
 run() (*PoolScanner method*), 281  
 run() (*PrintKey method*), 252  
 run() (*ProcDump method*), 283  
 run() (*Psaux method*), 237  
 run() (*PsList method*), 218, 239, 285  
 run() (*PsScan method*), 287  
 run() (*PsTree method*), 220, 241, 289  
 run() (*SSDT method*), 291  
 run() (*Statistics method*), 255  
 run() (*Strings method*), 293  
 run() (*SymlinkScan method*), 295  
 run() (*Tasks method*), 242  
 run() (*Timeliner method*), 307  
 run() (*UserAssist method*), 253  
 run() (*VadDump method*), 296  
 run() (*VadInfo method*), 298  
 run() (*VerInfo method*), 300  
 run() (*VirtMap method*), 302  
 run() (*VolShell method*), 23  
 run() (*Volshell method*), 25, 28, 30, 32

## S

sanitize\_name() (*TreeGrid static method*), 116, 308  
 save\_banners() (*LinuxBannerCache class method*), 39

save\_banners() (*MacBannerCache class method*), 42  
 save\_banners() (*SymbolBannerCache class method*), 52  
 scan() (*BufferDataLayer method*), 157  
 scan() (*DataLayerInterface method*), 104  
 scan() (*FileLayer method*), 159  
 scan() (*in module volatility.framework.automagic.pdbscan*), 48  
 scan() (*Intel method*), 130  
 scan() (*Intel32e method*), 133  
 scan() (*IntelPAE method*), 135  
 scan() (*LimeLayer method*), 147  
 scan() (*LinearlyMappedLayer method*), 149  
 scan() (*PdbMSFStream method*), 152  
 scan() (*PdbMultiStreamFormat method*), 154  
 scan() (*RegistryHive method*), 162  
 scan() (*SegmentedLayer method*), 165  
 scan() (*TranslationLayerInterface method*), 108  
 scan() (*VmwareLayer method*), 167  
 scan() (*WindowsCrashDump32Layer method*), 128  
 scan() (*WindowsIntel method*), 137  
 scan() (*WindowsIntel32e method*), 139  
 scan() (*WindowsIntelPAE method*), 142  
 scan() (*WindowsMixin method*), 144  
 scan\_drivers() (*DriverScan class method*), 263  
 scan\_files() (*FileScan class method*), 264  
 scan\_hives() (*HiveScan class method*), 250  
 scan\_modules() (*ModScan class method*), 274  
 scan\_mutants() (*MutantScan class method*), 277  
 scan\_processes() (*PsScan class method*), 287  
 scan\_symlinks() (*SymlinkScan class method*), 295  
 scannable\_sections() (*VirtMap class method*), 302  
 ScannerInterface (*class in volatility.framework.interfaces.layers*), 106  
 search() (*MultiRegexp method*), 126  
 second\_pass() (*DtbSelfRef32bit method*), 54  
 second\_pass() (*DtbSelfRef64bit method*), 54  
 second\_pass() (*DtbSelfReferential method*), 54  
 second\_pass() (*DtbTest method*), 55  
 second\_pass() (*DtbTest32bit method*), 55  
 second\_pass() (*DtbTest64bit method*), 55  
 second\_pass() (*DtbTestPae method*), 56  
 SegmentedLayer (*class in volatility.framework.layers.segmented*), 163  
 separator() (*HierarchicalDict property*), 96  
 SERVICE\_HEADER (*class in volatility.framework.symbols.windows.extensions.services*), 394  
 SERVICE\_HEADER.VolTemplateProxy (*class in volatility.framework.symbols.windows.extensions.services*), 394

SERVICE\_RECORD (class in volatility.framework.symbols.windows.extensions.services), 395

SERVICE\_RECORD.VolTemplateProxy (class in volatility.framework.symbols.windows.extensions.services), 395

set\_file\_consumer() (Bash method), 205, 222

set\_file\_consumer() (Certificates method), 246

set\_file\_consumer() (Check\_afinfo method), 207

set\_file\_consumer() (Check\_syscall method), 209, 224, 244

set\_file\_consumer() (Check\_sysctl method), 225

set\_file\_consumer() (Check\_trap\_table method), 227

set\_file\_consumer() (CmdLine method), 257

set\_file\_consumer() (ConfigWriter method), 303

set\_file\_consumer() (DllDump method), 258

set\_file\_consumer() (DllList method), 260

set\_file\_consumer() (DriverIrp method), 261

set\_file\_consumer() (DriverScan method), 263

set\_file\_consumer() (Elfs method), 210

set\_file\_consumer() (FileScan method), 265

set\_file\_consumer() (Handles method), 267

set\_file\_consumer() (HiveList method), 248

set\_file\_consumer() (HiveScan method), 250

set\_file\_consumer() (Ifconfig method), 228

set\_file\_consumer() (Info method), 268

set\_file\_consumer() (LayerWriter method), 305

set\_file\_consumer() (Lsmode method), 212, 230

set\_file\_consumer() (Lsof method), 213

set\_file\_consumer() (Lsof method), 231

set\_file\_consumer() (Malfind method), 215, 233, 270

set\_file\_consumer() (Maps method), 216, 236

set\_file\_consumer() (ModDump method), 272

set\_file\_consumer() (ModScan method), 274

set\_file\_consumer() (Modules method), 276

set\_file\_consumer() (MutantScan method), 278

set\_file\_consumer() (Netstat method), 234

set\_file\_consumer() (PluginInterface method), 114

set\_file\_consumer() (PoolScanner method), 281

set\_file\_consumer() (PrintKey method), 252

set\_file\_consumer() (ProcDump method), 283

set\_file\_consumer() (Psaux method), 238

set\_file\_consumer() (PsList method), 218, 239, 286

set\_file\_consumer() (PsScan method), 287

set\_file\_consumer() (PsTree method), 220, 241, 290

set\_file\_consumer() (SSDT method), 291

set\_file\_consumer() (Statistics method), 255

set\_file\_consumer() (Strings method), 293

set\_file\_consumer() (SymlinkScan method), 295

set\_file\_consumer() (Tasks method), 243

set\_file\_consumer() (Timeliner method), 307

set\_file\_consumer() (UserAssist method), 254

set\_file\_consumer() (VadDump method), 297

set\_file\_consumer() (VadInfo method), 298

set\_file\_consumer() (VerInfo method), 300

set\_file\_consumer() (VirtMap method), 302

set\_file\_consumer() (Volshell method), 25, 28, 30, 32

set\_kernel\_virtual\_offset() (KernelPDB-Scanner method), 47

set\_type\_class() (BaseSymbolTableInterface method), 119

set\_type\_class() (BashIntermedSymbols method), 341

set\_type\_class() (IntermediateSymbolTable method), 404

set\_type\_class() (ISFormatTable method), 400

set\_type\_class() (LinuxKernelIntermedSymbols method), 321

set\_type\_class() (MacKernelIntermedSymbols method), 344

set\_type\_class() (NativeTable method), 423

set\_type\_class() (NativeTableInterface method), 120

set\_type\_class() (SymbolTableInterface method), 124

set\_type\_class() (Version1Format method), 406

set\_type\_class() (Version2Format method), 408

set\_type\_class() (Version3Format method), 411

set\_type\_class() (Version4Format method), 413

set\_type\_class() (Version5Format method), 416

set\_type\_class() (Version6Format method), 418

set\_type\_class() (Version7Format method), 421

set\_type\_class() (WindowsKernelIntermedSymbols method), 359

SIGNATURE (WindowsCrashDump32Layer attribute), 126

signed() (DataFormatInfo property), 182

SimpleTypeRequirement (class in volatility.framework.interfaces.configuration), 98

size() (AggregateType.VolTemplateProxy class method), 169

size() (Array.VolTemplateProxy class method), 170

size() (BitField.VolTemplateProxy class method), 171

size() (Boolean.VolTemplateProxy class method), 173

size() (Bytes.VolTemplateProxy class method), 175

size() (Char.VolTemplateProxy class method), 179

size() (ClassType.VolTemplateProxy class method), 181

size() (CM\_KEY\_BODY.VolTemplateProxy class method), 388

size() (CM\_KEY\_NODE.VolTemplateProxy class



method), 389

size() (CM\_KEY\_VALUE.VolTemplateProxy class method), 391

size() (CMHIVE.VolTemplateProxy class method), 387

size() (dentry.VolTemplateProxy class method), 322

size() (DEVICE\_OBJECT.VolTemplateProxy class method), 360

size() (DRIVER\_OBJECT.VolTemplateProxy class method), 362

size() (Enumeration.VolTemplateProxy class method), 183

size() (EPROCESS.VolTemplateProxy class method), 363

size() (ETHREAD.VolTemplateProxy class method), 365

size() (EX\_FAST\_REF.VolTemplateProxy class method), 366

size() (ExecutiveObject.VolTemplateProxy class method), 367

size() (FILE\_OBJECT.VolTemplateProxy class method), 368

size() (fileglob.VolTemplateProxy class method), 345

size() (files\_struct.VolTemplateProxy class method), 323

size() (Float.VolTemplateProxy class method), 185

size() (fs\_struct.VolTemplateProxy class method), 325

size() (Function.VolTemplateProxy class method), 187

size() (GenericIntelProcess.VolTemplateProxy class method), 319

size() (hist\_entry.VolTemplateProxy class method), 338

size() (HMAP\_ENTRY.VolTemplateProxy class method), 392

size() (ifnet.VolTemplateProxy class method), 346

size() (IMAGE\_DOS\_HEADER.VolTemplateProxy class method), 384

size() (IMAGE\_NT\_HEADERS.VolTemplateProxy class method), 385

size() (inpcb.VolTemplateProxy class method), 347

size() (Integer.VolTemplateProxy class method), 188

size() (KDDEBUGGER\_DATA64.VolTemplateProxy class method), 382

size() (KMUTANT.VolTemplateProxy class method), 370

size() (KSYSTEM\_TIME.VolTemplateProxy class method), 371

size() (LIST\_ENTRY.VolTemplateProxy class method), 372

size() (list\_head.VolTemplateProxy class method), 326

size() (mm\_struct.VolTemplateProxy class method), 327

size() (MMVAD.VolTemplateProxy class method), 373

size() (MMVAD\_SHORT.VolTemplateProxy class method), 375

size() (module.VolTemplateProxy class method), 328

size() (mount.VolTemplateProxy class method), 329

size() (OBJECT\_HEADER.VolTemplateProxy class method), 377

size() (OBJECT\_SYMBOLIC\_LINK.VolTemplateProxy class method), 378

size() (ObjectInterface.VolTemplateProxy class method), 110

size() (ObjectTemplate property), 202

size() (Pointer.VolTemplateProxy class method), 190

size() (POOL\_HEADER.VolTemplateProxy class method), 380

size() (PrimitiveObject.VolTemplateProxy class method), 192

size() (proc.VolTemplateProxy class method), 348

size() (qstr.VolTemplateProxy class method), 331

size() (queue\_entry.VolTemplateProxy class method), 349

size() (ReferenceTemplate property), 203

size() (SERVICE\_HEADER.VolTemplateProxy class method), 394

size() (SERVICE\_RECORD.VolTemplateProxy class method), 395

size() (SizedModule property), 86

size() (sockaddr.VolTemplateProxy class method), 351

size() (sockaddr\_dl.VolTemplateProxy class method), 352

size() (socket.VolTemplateProxy class method), 353

size() (String.VolTemplateProxy class method), 193

size() (struct\_file.VolTemplateProxy class method), 332

size() (StructType.VolTemplateProxy class method), 198

size() (super\_block.VolTemplateProxy class method), 333

size() (SymbolSpace.UnresolvedTemplate property), 316

size() (task\_struct.VolTemplateProxy class method), 334

size() (Template property), 112

size() (UNICODE\_STRING.VolTemplateProxy class method), 381

size() (UnionType.VolTemplateProxy class method), 200

size() (v fsmount.VolTemplateProxy class method), 336

size() (vm\_area\_struct.VolTemplateProxy class method), 337

size() (vm\_map\_entry.VolTemplateProxy class method), 354

size() (vm\_map\_object.VolTemplateProxy class method), 355

size() (vnode.VolTemplateProxy class method), 357

size() (Void.VolTemplateProxy class method), 201

SizedModule (class in volatility.framework.contexts), 84

sockaddr (class in volatility.framework.symbols.mac.extensions), 350

sockaddr.VolTemplateProxy (class in volatility.framework.symbols.mac.extensions), 350

sockaddr\_dl (class in volatility.framework.symbols.mac.extensions), 351

sockaddr\_dl () (ifnet method), 346

sockaddr\_dl.VolTemplateProxy (class in volatility.framework.symbols.mac.extensions), 351

socket (class in volatility.framework.symbols.mac.extensions), 352

socket.VolTemplateProxy (class in volatility.framework.symbols.mac.extensions), 353

splice () (HierarchicalDict method), 96

split () (Bytes method), 178

split () (HexBytes method), 315

split () (String method), 197

splitlines () (Bytes method), 178

splitlines () (HexBytes method), 315

splitlines () (String method), 197

SSDT (class in volatility.plugins.windows.ssd), 290

stack () (LayerStacker method), 50

stack () (LimeStacker class method), 147

stack () (LintelStacker class method), 37

stack () (MacintelStacker class method), 44

stack () (StackerLayerInterface class method), 88

stack () (VmwareStacker class method), 168

stack () (WindowsCrashDump32Stacker class method), 128

stack () (WintelStacker class method), 59

stack\_order (LimeStacker attribute), 148

stack\_order (LintelStacker attribute), 37

stack\_order (MacintelStacker attribute), 44

stack\_order (StackerLayerInterface attribute), 88

stack\_order (VmwareStacker attribute), 168

stack\_order (WindowsCrashDump32Stacker attribute), 129

stack\_order (WintelStacker attribute), 59

StackerLayerInterface (class in volatility.framework.interfaces.automagic), 88

startswith () (Bytes method), 178

startswith () (HexBytes method), 315

startswith () (String method), 197

Statistics (class in volatility.plugins.windows.statistics), 254

String (class in volatility.framework.objects), 193

String () (UNICODE\_STRING property), 381

String.VolTemplateProxy (class in volatility.framework.objects), 193

StringRequirement (class in volatility.framework.configuration.requirements), 73

Strings (class in volatility.plugins.windows.strings), 292

strip () (Bytes method), 178

strip () (HexBytes method), 315

strip () (String method), 197

struct\_file (class in volatility.framework.symbols.linux.extensions), 331

struct\_file.VolTemplateProxy (class in volatility.framework.symbols.linux.extensions), 331

StructType (class in volatility.framework.objects), 198

StructType.VolTemplateProxy (class in volatility.framework.objects), 198

structure (Intel attribute), 131

structure (Intel32e attribute), 133

structure (IntelPAE attribute), 135

structure (WindowsIntel attribute), 137

structure (WindowsIntel32e attribute), 140

structure (WindowsIntelPAE attribute), 142

structure (WindowsMixin attribute), 144

super\_block (class in volatility.framework.symbols.linux.extensions), 332

super\_block.VolTemplateProxy (class in volatility.framework.symbols.linux.extensions), 333

swapcase () (Bytes method), 178

swapcase () (HexBytes method), 315

swapcase () (String method), 197

SwappedInvalidAddressException, 424

SYMBOL (SymbolType attribute), 318

SYMBOL\_BASEPATHS (in module volatility.framework.constants), 80

symbol\_class (LinuxSymbolFinder attribute), 40

symbol\_class (MacSymbolFinder attribute), 44

symbol\_class (SymbolFinder attribute), 53

symbol\_name (LinuxBannerCache attribute), 39

symbol\_name (MacBannerCache attribute), 42

symbol\_name (SymbolBannerCache attribute), 52

symbol\_space () (Context property), 82

symbol\_space () (ContextInterface property), 101

symbol\_table\_is\_64bit () (in module volatility.framework.symbols), 318

SymbolBannerCache (class in volatility.framework.automagic.symbol\_cache), 50

SymbolError, 425

SymbolFinder (class in volatility.framework.automagic.symbol\_finder), 52

- SymbolInterface (class in volatility.framework.interfaces.symbols), 121
  - symbols() (BaseSymbolTableInterface property), 119
  - symbols() (BashIntermedSymbols property), 341
  - symbols() (IntermediateSymbolTable property), 404
  - symbols() (ISFormatTable property), 401
  - symbols() (LinuxKernelIntermedSymbols property), 321
  - symbols() (MacKernelIntermedSymbols property), 344
  - symbols() (NativeTable property), 423
  - symbols() (NativeTableInterface property), 120
  - symbols() (SymbolTableInterface property), 124
  - symbols() (Version1Format property), 406
  - symbols() (Version2Format property), 408
  - symbols() (Version3Format property), 411
  - symbols() (Version4Format property), 413
  - symbols() (Version5Format property), 416
  - symbols() (Version6Format property), 418
  - symbols() (Version7Format property), 421
  - symbols() (WindowsKernelIntermedSymbols property), 359
  - SymbolSpace (class in volatility.framework.symbols), 316
  - SymbolSpace.UnresolvedTemplate (class in volatility.framework.symbols), 316
  - SymbolSpaceError, 425
  - SymbolSpaceInterface (class in volatility.framework.interfaces.symbols), 121
  - SymbolTableInterface (class in volatility.framework.interfaces.symbols), 122
  - SymbolTableRequirement (class in volatility.framework.configuration.requirements), 75
  - SymbolType (class in volatility.framework.symbols), 318
  - SymlinkScan (class in volatility.plugins.windows.symlinkscan), 294
- ## T
- task\_struct (class in volatility.framework.symbols.linux.extensions), 334
  - task\_struct.VolTemplateProxy (class in volatility.framework.symbols.linux.extensions), 334
  - Tasks (class in volatility.plugins.mac.tasks), 241
  - Template (class in volatility.framework.interfaces.objects), 111
  - tests (PageMapScanner attribute), 56
  - tests (WintelHelper attribute), 59
  - thread\_safe (BytesScanner attribute), 125
  - thread\_safe (MultiStringScanner attribute), 125
  - thread\_safe (PageMapScanner attribute), 56
  - thread\_safe (PdbSignatureScanner attribute), 48
  - thread\_safe (PoolHeaderScanner attribute), 278
  - thread\_safe (RegExScanner attribute), 125
  - thread\_safe (ScannerInterface attribute), 107
  - Threading (Parallelism attribute), 80
  - Timeliner (class in volatility.plugins.timeliner), 306
  - TimeLinerInterface (class in volatility.plugins.timeliner), 305
  - TimeLinerType (class in volatility.plugins.timeliner), 306
  - title() (Bytes method), 178
  - title() (HexBytes method), 315
  - title() (String method), 197
  - to\_bytes() (Bin method), 311
  - to\_bytes() (BitField method), 172
  - to\_bytes() (Boolean method), 174
  - to\_bytes() (Char method), 180
  - to\_bytes() (Enumeration method), 184
  - to\_bytes() (Hex method), 312
  - to\_bytes() (Integer method), 189
  - to\_bytes() (Pointer method), 191
  - to\_list() (LIST\_ENTRY method), 373
  - to\_list() (list\_head method), 326
  - translate() (Bytes method), 178
  - translate() (HexBytes method), 315
  - translate() (Intel method), 131
  - translate() (Intel32e method), 133
  - translate() (IntelPAE method), 135
  - translate() (LimeLayer method), 147
  - translate() (LinearlyMappedLayer method), 150
  - translate() (PdbMSFStream method), 152
  - translate() (PdbMultiStreamFormat method), 155
  - translate() (RegistryHive method), 162
  - translate() (SegmentedLayer method), 165
  - translate() (String method), 198
  - translate() (VmwareLayer method), 168
  - translate() (WindowsCrashDump32Layer method), 128
  - translate() (WindowsIntel method), 138
  - translate() (WindowsIntel32e method), 140
  - translate() (WindowsIntelPAE method), 142
  - translate() (WindowsMixin method), 144
  - TranslationLayerInterface (class in volatility.framework.interfaces.layers), 107
  - TranslationLayerRequirement (class in volatility.framework.configuration.requirements), 76
  - traverse() (MMVAD method), 374
  - traverse() (MMVAD\_SHORT method), 376
  - traverse() (SERVICE\_RECORD method), 396
  - TreeGrid (class in volatility.framework.interfaces.renderers), 115
  - TreeGrid (class in volatility.framework.renderers), 307
  - TreeNode (class in volatility.framework.interfaces.renderers), 117

- TreeNode (class in volatility.framework.renderers), 309  
TYPE (SymbolType attribute), 318  
type () (Column property), 115  
type () (SymbolInterface property), 121  
type\_name () (SymbolInterface property), 121  
types () (BaseSymbolTableInterface property), 119  
types () (BashIntermedSymbols property), 341  
types () (IntermediateSymbolTable property), 404  
types () (ISFormatTable property), 401  
types () (LinuxKernelIntermedSymbols property), 321  
types () (MacKernelIntermedSymbols property), 344  
types () (NativeTable property), 423  
types () (NativeTableInterface property), 121  
types () (SymbolTableInterface property), 124  
types () (Version1Format property), 406  
types () (Version2Format property), 408  
types () (Version3Format property), 411  
types () (Version4Format property), 413  
types () (Version5Format property), 416  
types () (Version6Format property), 419  
types () (Version7Format property), 421  
types () (WindowsKernelIntermedSymbols property), 359
- ## U
- UNICODE\_STRING (class in volatility.framework.symbols.windows.extensions), 380  
UNICODE\_STRING.VolTemplateProxy (class in volatility.framework.symbols.windows.extensions), 381  
UnionType (class in volatility.framework.objects), 199  
UnionType.VolTemplateProxy (class in volatility.framework.objects), 199  
unixtime\_to\_datetime () (in module volatility.framework.renderers.conversion), 310  
UnparsableValue (class in volatility.framework.renderers), 309  
UnreadableValue (class in volatility.framework.renderers), 310  
unsatisfied () (AutomagicInterface class method), 88  
unsatisfied () (Bash class method), 206, 222  
unsatisfied () (BashIntermedSymbols class method), 341  
unsatisfied () (BooleanRequirement method), 61  
unsatisfied () (BufferDataLayer class method), 157  
unsatisfied () (BytesRequirement method), 62  
unsatisfied () (Certificates class method), 246  
unsatisfied () (Check\_afinfo class method), 207  
unsatisfied () (Check\_syscall class method), 209, 224, 244  
unsatisfied () (Check\_sysctl class method), 225  
unsatisfied () (Check\_trap\_table class method), 227  
unsatisfied () (ChoiceRequirement method), 63  
unsatisfied () (ClassRequirement method), 90  
unsatisfied () (CmdLine class method), 257  
unsatisfied () (ComplexListRequirement method), 65  
unsatisfied () (ConfigurableInterface class method), 91  
unsatisfied () (ConfigurableRequirementInterface method), 92  
unsatisfied () (ConfigWriter class method), 303  
unsatisfied () (ConstructableRequirementInterface method), 94  
unsatisfied () (ConstructionMagic class method), 37  
unsatisfied () (DataLayerInterface class method), 105  
unsatisfied () (DllDump class method), 258  
unsatisfied () (DllList class method), 260  
unsatisfied () (DriverIrp class method), 261  
unsatisfied () (DriverScan class method), 263  
unsatisfied () (Elfs class method), 210  
unsatisfied () (FileLayer class method), 159  
unsatisfied () (FileScan class method), 265  
unsatisfied () (Handles class method), 267  
unsatisfied () (HiveList class method), 249  
unsatisfied () (HiveScan class method), 250  
unsatisfied () (Ifconfig class method), 228  
unsatisfied () (Info class method), 268  
unsatisfied () (Intel class method), 131  
unsatisfied () (Intel32e class method), 133  
unsatisfied () (IntelPAE class method), 135  
unsatisfied () (IntermediateSymbolTable class method), 404  
unsatisfied () (IntRequirement method), 66  
unsatisfied () (ISFormatTable class method), 401  
unsatisfied () (KernelPDBScanner class method), 47  
unsatisfied () (LayerListRequirement method), 68  
unsatisfied () (LayerStacker class method), 50  
unsatisfied () (LayerWriter class method), 305  
unsatisfied () (LimeLayer class method), 147  
unsatisfied () (LinearlyMappedLayer class method), 150  
unsatisfied () (LinuxBannerCache class method), 39  
unsatisfied () (LinuxKernelIntermedSymbols class method), 321  
unsatisfied () (LinuxSymbolFinder class method), 40  
unsatisfied () (ListRequirement method), 70  
unsatisfied () (Lsmmod class method), 212, 230  
unsatisfied () (Lsof class method), 213



[unsatisfied\(\)](#) (*Isof* class method), 232  
[unsatisfied\(\)](#) (*MacBannerCache* class method), 42  
[unsatisfied\(\)](#) (*MacKernelIntermedSymbols* class method), 344  
[unsatisfied\(\)](#) (*MacSymbolFinder* class method), 44  
[unsatisfied\(\)](#) (*Malfind* class method), 215, 233, 270  
[unsatisfied\(\)](#) (*Maps* class method), 217, 236  
[unsatisfied\(\)](#) (*ModDump* class method), 272  
[unsatisfied\(\)](#) (*ModScan* class method), 274  
[unsatisfied\(\)](#) (*Modules* class method), 276  
[unsatisfied\(\)](#) (*MultiRequirement* method), 71  
[unsatisfied\(\)](#) (*MutantScan* class method), 278  
[unsatisfied\(\)](#) (*Netstat* class method), 235  
[unsatisfied\(\)](#) (*PdbMSFStream* class method), 152  
[unsatisfied\(\)](#) (*PdbMultiStreamFormat* class method), 155  
[unsatisfied\(\)](#) (*PluginInterface* class method), 114  
[unsatisfied\(\)](#) (*PluginRequirement* method), 73  
[unsatisfied\(\)](#) (*PoolScanner* class method), 281  
[unsatisfied\(\)](#) (*PrintKey* class method), 252  
[unsatisfied\(\)](#) (*ProcDump* class method), 283  
[unsatisfied\(\)](#) (*Psaux* class method), 238  
[unsatisfied\(\)](#) (*PsList* class method), 218, 239, 286  
[unsatisfied\(\)](#) (*PsScan* class method), 288  
[unsatisfied\(\)](#) (*PsTree* class method), 220, 241, 290  
[unsatisfied\(\)](#) (*RegistryHive* class method), 162  
[unsatisfied\(\)](#) (*RequirementInterface* method), 97  
[unsatisfied\(\)](#) (*SegmentedLayer* class method), 165  
[unsatisfied\(\)](#) (*SimpleTypeRequirement* method), 99  
[unsatisfied\(\)](#) (*SSDT* class method), 292  
[unsatisfied\(\)](#) (*Statistics* class method), 255  
[unsatisfied\(\)](#) (*StringRequirement* method), 74  
[unsatisfied\(\)](#) (*Strings* class method), 293  
[unsatisfied\(\)](#) (*SymbolBannerCache* class method), 52  
[unsatisfied\(\)](#) (*SymbolFinder* class method), 53  
[unsatisfied\(\)](#) (*SymbolTableInterface* class method), 124  
[unsatisfied\(\)](#) (*SymbolTableRequirement* method), 76  
[unsatisfied\(\)](#) (*SymlinkScan* class method), 295  
[unsatisfied\(\)](#) (*Tasks* class method), 243  
[unsatisfied\(\)](#) (*Timeliner* class method), 307  
[unsatisfied\(\)](#) (*TranslationLayerInterface* class method), 109  
[unsatisfied\(\)](#) (*TranslationLayerRequirement* method), 78  
[unsatisfied\(\)](#) (*URIRequirement* method), 79  
[unsatisfied\(\)](#) (*UserAssist* class method), 254  
[unsatisfied\(\)](#) (*VadDump* class method), 297  
[unsatisfied\(\)](#) (*VadInfo* class method), 298  
[unsatisfied\(\)](#) (*VerInfo* class method), 300  
[unsatisfied\(\)](#) (*Version1Format* class method), 406  
[unsatisfied\(\)](#) (*Version2Format* class method), 409  
[unsatisfied\(\)](#) (*Version3Format* class method), 411  
[unsatisfied\(\)](#) (*Version4Format* class method), 414  
[unsatisfied\(\)](#) (*Version5Format* class method), 416  
[unsatisfied\(\)](#) (*Version6Format* class method), 419  
[unsatisfied\(\)](#) (*Version7Format* class method), 421  
[unsatisfied\(\)](#) (*VirtMap* class method), 302  
[unsatisfied\(\)](#) (*VmwareLayer* class method), 168  
[unsatisfied\(\)](#) (*Volshell* class method), 25, 28, 30, 32  
[unsatisfied\(\)](#) (*WindowsCrashDump32Layer* class method), 128  
[unsatisfied\(\)](#) (*WindowsIntel* class method), 138  
[unsatisfied\(\)](#) (*WindowsIntel32e* class method), 140  
[unsatisfied\(\)](#) (*WindowsIntelPAE* class method), 142  
[unsatisfied\(\)](#) (*WindowsKernelIntermedSymbols* class method), 360  
[unsatisfied\(\)](#) (*WindowsMixin* class method), 144  
[unsatisfied\(\)](#) (*WinSwapLayers* class method), 57  
[unsatisfied\(\)](#) (*WintelHelper* class method), 59  
[unsatisfied\\_children\(\)](#) (*BooleanRequirement* method), 61  
[unsatisfied\\_children\(\)](#) (*BytesRequirement* method), 62  
[unsatisfied\\_children\(\)](#) (*ChoiceRequirement* method), 63  
[unsatisfied\\_children\(\)](#) (*ClassRequirement* method), 90  
[unsatisfied\\_children\(\)](#) (*ComplexListRequirement* method), 65  
[unsatisfied\\_children\(\)](#) (*ConfigurableRequirementInterface* method), 93  
[unsatisfied\\_children\(\)](#) (*ConstructableRequirementInterface* method), 94  
[unsatisfied\\_children\(\)](#) (*IntRequirement* method), 67  
[unsatisfied\\_children\(\)](#) (*LayerListRequirement* method), 68  
[unsatisfied\\_children\(\)](#) (*ListRequirement* method), 70  
[unsatisfied\\_children\(\)](#) (*MultiRequirement* method), 72  
[unsatisfied\\_children\(\)](#) (*PluginRequirement* method), 73  
[unsatisfied\\_children\(\)](#) (*RequirementInterface* method), 97  
[unsatisfied\\_children\(\)](#) (*SimpleTypeRequirement* method), 99  
[unsatisfied\\_children\(\)](#) (*StringRequirement* method), 74  
[unsatisfied\\_children\(\)](#) (*SymbolTableRequirement* method), 76  
[unsatisfied\\_children\(\)](#) (*TranslationLayerRequirement* method), 78

`unsatisfied_children()` (*URIRequirement* method), 79

`UnsatisfiedException`, 425

`update_vol()` (*ObjectTemplate* method), 202

`update_vol()` (*ReferenceTemplate* method), 203

`update_vol()` (*SymbolSpace.UnresolvedTemplate* method), 317

`update_vol()` (*Template* method), 112

`upper()` (*Bytes* method), 179

`upper()` (*HexBytes* method), 315

`upper()` (*String* method), 198

`URIRequirement` (class in *volatility.framework.configuration.requirements*), 78

`UserAssist` (class in *volatility.plugins.windows.registry.userassist*), 252

## V

`VadDump` (class in *volatility.plugins.windows.vaddump*), 295

`VadInfo` (class in *volatility.plugins.windows.vadinfo*), 297

`valid()` (in module *volatility.schemas*), 426

`validate()` (in module *volatility.schemas*), 426

`VALIDDUMP` (*WindowsCrashDump32Layer* attribute), 126

`values()` (*HierarchicalDict* method), 96

`values()` (*LayerContainer* method), 106

`values()` (*ObjectInformation* method), 110

`values()` (*ReadOnlyMapping* method), 111

`values()` (*SymbolSpace* method), 318

`values()` (*SymbolSpaceInterface* method), 122

`values()` (*TreeGrid* method), 116, 309

`values()` (*TreeNode* property), 118, 309

`VerInfo` (class in *volatility.plugins.windows.verinfo*), 299

`version` (*Bash* attribute), 206, 222

`version` (*Certificates* attribute), 247

`version` (*Check\_afinfo* attribute), 207

`version` (*Check\_syscall* attribute), 209, 224, 244

`version` (*Check\_sysctl* attribute), 225

`version` (*Check\_trap\_table* attribute), 227

`version` (*CmdLine* attribute), 257

`version` (*ConfigWriter* attribute), 304

`version` (*DllDump* attribute), 258

`version` (*DllList* attribute), 260

`version` (*DriverIrp* attribute), 262

`version` (*DriverScan* attribute), 263

`version` (*Elfs* attribute), 210

`version` (*FileScan* attribute), 265

`version` (*Handles* attribute), 267

`version` (*HiveList* attribute), 249

`version` (*HiveScan* attribute), 250

`version` (*Ifconfig* attribute), 229

`version` (*Info* attribute), 269

`version` (*ISFormatTable* attribute), 401

`version` (*LayerWriter* attribute), 305

`VERSION` (*LimeLayer* attribute), 145

`version` (*Lsmode* attribute), 212, 230

`version` (*Lsof* attribute), 214

`version` (*Lsof* attribute), 232

`version` (*Malfind* attribute), 215, 233, 271

`version` (*Maps* attribute), 217, 236

`version` (*ModDump* attribute), 273

`version` (*ModScan* attribute), 274

`version` (*Modules* attribute), 276

`version` (*MutantScan* attribute), 278

`version` (*Netstat* attribute), 235

`version` (*PluginInterface* attribute), 114

`version` (*PoolScanner* attribute), 281

`version` (*PrintKey* attribute), 252

`version` (*ProcDump* attribute), 284

`version` (*Psaux* attribute), 238

`version` (*PsList* attribute), 219, 240, 286

`version` (*PsScan* attribute), 288

`version` (*PsTree* attribute), 221, 241, 290

`version` (*SSDT* attribute), 292

`version` (*Statistics* attribute), 255

`version` (*Strings* attribute), 293

`version` (*SymlinkScan* attribute), 295

`version` (*Tasks* attribute), 243

`version` (*Timeliner* attribute), 307

`version` (*UserAssist* attribute), 254

`version` (*VadDump* attribute), 297

`version` (*VadInfo* attribute), 299

`version` (*VerInfo* attribute), 300

`version` (*Version1Format* attribute), 406

`version` (*Version2Format* attribute), 409

`version` (*Version3Format* attribute), 411

`version` (*Version4Format* attribute), 414

`version` (*Version5Format* attribute), 416

`version` (*Version6Format* attribute), 419

`version` (*Version7Format* attribute), 421

`version` (*VirtMap* attribute), 302

`version` (*Volshell* attribute), 26, 28, 30, 32

`Version1Format` (class in *volatility.framework.symbols.intermed*), 404

`Version2Format` (class in *volatility.framework.symbols.intermed*), 406

`Version3Format` (class in *volatility.framework.symbols.intermed*), 409

`Version4Format` (class in *volatility.framework.symbols.intermed*), 411

`Version5Format` (class in *volatility.framework.symbols.intermed*), 414

`Version6Format` (class in *volatility.framework.symbols.intermed*), 416

Version7Format (class in volatility.framework.symbols.intermed), 419

vfsmount (class in volatility.framework.symbols.linux.extensions), 335

vfsmount.VolTemplateProxy (class in volatility.framework.symbols.linux.extensions), 335

VirtMap (class in volatility.plugins.windows.virtmap), 301

virtual\_to\_physical\_address() (LinuxUtilities class method), 41

virtual\_to\_physical\_address() (MacUtilities class method), 44

visit() (TreeGrid method), 117, 309

visit\_nodes() (RegistryHive method), 162

vm\_area\_struct (class in volatility.framework.symbols.linux.extensions), 336

vm\_area\_struct.VolTemplateProxy (class in volatility.framework.symbols.linux.extensions), 336

vm\_map\_entry (class in volatility.framework.symbols.mac.extensions), 354

vm\_map\_entry.VolTemplateProxy (class in volatility.framework.symbols.mac.extensions), 354

vm\_map\_object (class in volatility.framework.symbols.mac.extensions), 355

vm\_map\_object.VolTemplateProxy (class in volatility.framework.symbols.mac.extensions), 355

VmwareLayer (class in volatility.framework.layers.vmware), 166

VmwareStacker (class in volatility.framework.layers.vmware), 168

vnode (class in volatility.framework.symbols.mac.extensions), 356

vnode.VolTemplateProxy (class in volatility.framework.symbols.mac.extensions), 356

Void (class in volatility.framework.objects), 200

Void.VolTemplateProxy (class in volatility.framework.objects), 200

vol() (AggregateType property), 169

vol() (Array property), 171

vol() (BitField property), 172

vol() (Boolean property), 174

vol() (Bytes property), 179

vol() (Char property), 181

vol() (ClassType property), 182

vol() (CM\_KEY\_BODY property), 389

vol() (CM\_KEY\_NODE property), 390

vol() (CM\_KEY\_VALUE property), 391

vol() (CMHIVE property), 387

vol() (dentry property), 323

vol() (DEVICE\_OBJECT property), 361

vol() (DRIVER\_OBJECT property), 362

vol() (Enumeration property), 184

vol() (EPROCESS property), 364

vol() (ETHREAD property), 365

vol() (EX\_FAST\_REF property), 366

vol() (ExecutiveObject property), 368

vol() (FILE\_OBJECT property), 369

vol() (fileglob property), 345

vol() (files\_struct property), 324

vol() (Float property), 186

vol() (fs\_struct property), 325

vol() (Function property), 187

vol() (GenericIntelProcess property), 319

vol() (hist\_entry property), 339

vol() (HMAP\_ENTRY property), 392

vol() (ifnet property), 346

vol() (IMAGE\_DOS\_HEADER property), 385

vol() (IMAGE\_NT\_HEADERS property), 386

vol() (inpcb property), 347

vol() (Integer property), 189

vol() (KDDEBUGGER\_DATA64 property), 383

vol() (KMUTANT property), 370

vol() (KSYSTEM\_TIME property), 372

vol() (LIST\_ENTRY property), 373

vol() (list\_head property), 326

vol() (mm\_struct property), 328

vol() (MMVAD property), 374

vol() (MMVAD\_SHORT property), 376

vol() (module property), 329

vol() (mount property), 330

vol() (OBJECT\_HEADER property), 378

vol() (OBJECT\_SYMBOLIC\_LINK property), 379

vol() (ObjectInterface property), 111

vol() (ObjectTemplate property), 202

vol() (Pointer property), 191

vol() (POOL\_HEADER property), 380

vol() (PrimitiveObject property), 193

vol() (proc property), 349

vol() (qstr property), 331

vol() (queue\_entry property), 350

vol() (ReferenceTemplate property), 203

vol() (SERVICE\_HEADER property), 395

vol() (SERVICE\_RECORD property), 396

vol() (sockaddr property), 351

vol() (sockaddr\_dl property), 352

vol() (socket property), 353

vol() (String property), 198

vol() (struct\_file property), 332

vol() (StructType property), 199

vol() (super\_block property), 334

vol() (SymbolSpace.UnresolvedTemplate property), 317

vol() (task\_struct property), 335

`vol()` (*Template property*), 112  
`vol()` (*UNICODE\_STRING property*), 382  
`vol()` (*UnionType property*), 200  
`vol()` (*vfsmount property*), 336  
`vol()` (*vm\_area\_struct property*), 338  
`vol()` (*vm\_map\_entry property*), 355  
`vol()` (*vm\_map\_object property*), 356  
`vol()` (*vnode property*), 357  
`vol()` (*Void property*), 201  
`volatility` (*module*), 21  
`volatility.cli` (*module*), 21  
`volatility.cli.text_renderer` (*module*), 33  
`volatility.cli.volshell` (*module*), 23  
`volatility.cli.volshell.generic` (*module*), 23  
`volatility.cli.volshell.linux` (*module*), 26  
`volatility.cli.volshell.mac` (*module*), 28  
`volatility.cli.volshell.windows` (*module*), 30  
`volatility.framework` (*module*), 34  
`volatility.framework.automagic` (*module*), 35  
`volatility.framework.automagic.constructors` (*module*), 36  
`volatility.framework.automagic.linux` (*module*), 37  
`volatility.framework.automagic.mac` (*module*), 41  
`volatility.framework.automagic.pdbscan` (*module*), 44  
`volatility.framework.automagic.stacker` (*module*), 48  
`volatility.framework.automagic.symbol_cache` (*module*), 50  
`volatility.framework.automagic.symbol_finder` (*module*), 52  
`volatility.framework.automagic.windows` (*module*), 54  
`volatility.framework.configuration` (*module*), 59  
`volatility.framework.configuration.requirements` (*module*), 60  
`volatility.framework.constants` (*module*), 79  
`volatility.framework.constants.linux` (*module*), 80  
`volatility.framework.constants.windows` (*module*), 81  
`volatility.framework.contexts` (*module*), 81  
`volatility.framework.exceptions` (*module*), 424  
`volatility.framework.interfaces` (*module*), 86  
`volatility.framework.interfaces.automagic` (*module*), 86  
`volatility.framework.interfaces.configuration` (*module*), 89  
`volatility.framework.interfaces.context` (*module*), 99  
`volatility.framework.interfaces.layers` (*module*), 102  
`volatility.framework.interfaces.objects` (*module*), 109  
`volatility.framework.interfaces.plugins` (*module*), 112  
`volatility.framework.interfaces.renderers` (*module*), 114  
`volatility.framework.interfaces.symbols` (*module*), 118  
`volatility.framework.layers` (*module*), 125  
`volatility.framework.layers.crash` (*module*), 126  
`volatility.framework.layers.intel` (*module*), 129  
`volatility.framework.layers.lime` (*module*), 145  
`volatility.framework.layers.linear` (*module*), 148  
`volatility.framework.layers.msf` (*module*), 150  
`volatility.framework.layers.physical` (*module*), 155  
`volatility.framework.layers.registry` (*module*), 160  
`volatility.framework.layers.resources` (*module*), 163  
`volatility.framework.layers.scanners` (*module*), 125  
`volatility.framework.layers.scanners.multiregexp` (*module*), 125  
`volatility.framework.layers.segmented` (*module*), 163  
`volatility.framework.layers.vmware` (*module*), 166  
`volatility.framework.objects` (*module*), 168  
`volatility.framework.objects.templates` (*module*), 202  
`volatility.framework.objects.utility` (*module*), 203  
`volatility.framework.renderers` (*module*), 307  
`volatility.framework.renderers.conversion` (*module*), 310  
`volatility.framework.renderers.format_hints` (*module*), 310  
`volatility.framework.symbols` (*module*), 316  
`volatility.framework.symbols.generic` (*module*), 318



volatility.framework.symbols.intermed (module), 399	volatility.plugins.linux.proc (module), 215
volatility.framework.symbols.linux (mod- ule), 319	volatility.plugins.linux.pslist (module), 217
volatility.framework.symbols.linux.bash (module), 339	volatility.plugins.linux.pstree (module), 219
volatility.framework.symbols.linux.extensions (module), 322	volatility.plugins.mac (module), 221
volatility.framework.symbols.linux.extensions (module), 338	volatility.plugins.mac.bash (module), 221
volatility.framework.symbols.mac (mod- ule), 342	volatility.plugins.mac.check_syscall (module), 223
volatility.framework.symbols.mac.extensions (module), 344	volatility.plugins.mac.check_sysctl (module), 224
volatility.framework.symbols.metadata (module), 421	volatility.plugins.mac.check_trap_table (module), 226
volatility.framework.symbols.native (module), 422	volatility.plugins.mac.ifconfig (module), 227
volatility.framework.symbols.windows (module), 357	volatility.plugins.mac.lsmmod (module), 229
volatility.framework.symbols.windows.extensions (module), 360	volatility.plugins.mac.lsof (module), 230
volatility.framework.symbols.windows.extensions (module), 382	volatility.plugins.mac.malfind (module), 232
volatility.framework.symbols.windows.extensions (module), 383	volatility.plugins.mac.netstat (module), 233
volatility.framework.symbols.windows.extensions (module), 386	volatility.plugins.mac.proc_maps (mod- ule), 235
volatility.framework.symbols.windows.extensions (module), 394	volatility.plugins.mac.psaux (module), 236
volatility.framework.symbols.windows.pdbvoh (module), 397	volatility.plugins.mac.pslist (module), 238
volatility.framework.symbols.wrappers (module), 423	volatility.plugins.mac.registry (module), 240
volatility.plugins (module), 204	volatility.plugins.mac.services (module), 241
volatility.plugins.configwriter (module), 302	volatility.plugins.mac.taskmgr (module), 243
volatility.plugins.layerwriter (module), 304	volatility.plugins.timeliner (module), 305
volatility.plugins.linux (module), 204	volatility.plugins.windows (module), 245
volatility.plugins.linux.bash (module), 204	volatility.plugins.windows.cmdline (mod- ule), 256
volatility.plugins.linux.check_afinfo (module), 206	volatility.plugins.windows.dlldump (mod- ule), 257
volatility.plugins.linux.check_syscall (module), 207	volatility.plugins.windows.dlllist (mod- ule), 259
volatility.plugins.linux.elfs (module), 209	volatility.plugins.windows.driverirp (module), 260
volatility.plugins.linux.lsmmod (module), 210	volatility.plugins.windows.driverscan (module), 262
volatility.plugins.linux.lsof (module), 212	volatility.plugins.windows.filescan (module), 263
volatility.plugins.linux.malfind (mod- ule), 214	volatility.plugins.windows.handles (mod- ule), 265
	volatility.plugins.windows.info (module), 267
	volatility.plugins.windows.malfind (mod- ule), 269
	volatility.plugins.windows.moddump (mod- ule), 271

`volatility.plugins.windows.modscan` (*module*), 273  
`volatility.plugins.windows.modules` (*module*), 275  
`volatility.plugins.windows.mutantscan` (*module*), 276  
`volatility.plugins.windows.poolscanner` (*module*), 278  
`volatility.plugins.windows.procdump` (*module*), 282  
`volatility.plugins.windows.pslist` (*module*), 284  
`volatility.plugins.windows.psscan` (*module*), 286  
`volatility.plugins.windows.pstree` (*module*), 288  
`volatility.plugins.windows.registry` (*module*), 245  
`volatility.plugins.windows.registry.certificates` (*module*), 245  
`volatility.plugins.windows.registry.hivelist` (*module*), 247  
`volatility.plugins.windows.registry.hivescan` (*module*), 249  
`volatility.plugins.windows.registry.printkey` (*module*), 251  
`volatility.plugins.windows.registry.userassist` (*module*), 252  
`volatility.plugins.windows.ssd` (*module*), 290  
`volatility.plugins.windows.statistics` (*module*), 254  
`volatility.plugins.windows.strings` (*module*), 292  
`volatility.plugins.windows.symlinkscan` (*module*), 294  
`volatility.plugins.windows.vaddump` (*module*), 295  
`volatility.plugins.windows.vadinfo` (*module*), 297  
`volatility.plugins.windows.verinfo` (*module*), 299  
`volatility.plugins.windows.virtmap` (*module*), 301  
`volatility.schemas` (*module*), 425  
`volatility.symbols` (*module*), 426  
`VolatilityException`, 425  
`VolShell` (*class in volatility.cli.volshell*), 23  
`Volshell` (*class in volatility.cli.volshell.generic*), 24  
`Volshell` (*class in volatility.cli.volshell.linux*), 26  
`Volshell` (*class in volatility.cli.volshell.mac*), 28  
`Volshell` (*class in volatility.cli.volshell.windows*), 30

## W

`walk_list()` (*queue\_entry method*), 350  
`walk_tailq()` (*MacUtilities method*), 44  
`WarningFindSpec` (*class in volatility*), 21  
`WindowsCrashDump32FormatException`, 126  
`WindowsCrashDump32Layer` (*class in volatility.framework.layers.crash*), 126  
`WindowsCrashDump32Stacker` (*class in volatility.framework.layers.crash*), 128  
`WindowsIntel` (*class in volatility.framework.layers.intel*), 136  
`WindowsIntel32e` (*class in volatility.framework.layers.intel*), 138  
`WindowsIntelPAE` (*class in volatility.framework.layers.intel*), 140  
`WindowsKernelIntermedSymbols` (*class in volatility.framework.symbols.windows*), 357  
`WindowsMetadata` (*class in volatility.framework.symbols.metadata*), 422  
`WindowsMixin` (*class in volatility.framework.layers.intel*), 142  
`WinSwapLayers` (*class in volatility.framework.automagic.windows*), 56  
`WintelHelper` (*class in volatility.framework.automagic.windows*), 58  
`WintelStacker` (*class in volatility.framework.automagic.windows*), 59  
`wintime_to_datetime()` (*in module volatility.framework.renderers.conversion*), 310  
`with_traceback()` (*InvalidAddressException method*), 424  
`with_traceback()` (*LayerException method*), 424  
`with_traceback()` (*LimeFormatException method*), 145  
`with_traceback()` (*PagedInvalidAddressException method*), 424  
`with_traceback()` (*PluginRequirementException method*), 424  
`with_traceback()` (*PluginVersionException method*), 424  
`with_traceback()` (*RegistryFormatException method*), 160  
`with_traceback()` (*RegistryInvalidIndex method*), 163  
`with_traceback()` (*SwappedInvalidAddressException method*), 425  
`with_traceback()` (*SymbolError method*), 425  
`with_traceback()` (*SymbolSpaceError method*), 425  
`with_traceback()` (*UnsatisfiedException method*), 425  
`with_traceback()` (*VolatilityException method*), 425

with\_traceback() (WindowsCrash-Dump32FormatException method), 126  
 write() (AggregateType method), 169  
 write() (Array method), 171  
 write() (BitField method), 173  
 write() (Boolean method), 174  
 write() (BufferDataLayer method), 157  
 write() (Bytes method), 179  
 write() (Char method), 181  
 write() (ClassType method), 182  
 write() (CM\_KEY\_BODY method), 389  
 write() (CM\_KEY\_NODE method), 390  
 write() (CM\_KEY\_VALUE method), 391  
 write() (CMHIVE method), 387  
 write() (DataLayerInterface method), 105  
 write() (dentry method), 323  
 write() (DEVICE\_OBJECT method), 361  
 write() (DRIVER\_OBJECT method), 362  
 write() (Enumeration method), 184  
 write() (EPROCESS method), 364  
 write() (ETHREAD method), 365  
 write() (EX\_FAST\_REF method), 366  
 write() (ExecutiveObject method), 368  
 write() (FILE\_OBJECT method), 369  
 write() (fileglob method), 345  
 write() (FileLayer method), 160  
 write() (files\_struct method), 324  
 write() (Float method), 186  
 write() (fs\_struct method), 325  
 write() (Function method), 187  
 write() (GenericIntelProcess method), 319  
 write() (hist\_entry method), 339  
 write() (HMAP\_ENTRY method), 393  
 write() (ifnet method), 346  
 write() (IMAGE\_DOS\_HEADER method), 385  
 write() (IMAGE\_NT\_HEADERS method), 386  
 write() (inpcb method), 347  
 write() (Integer method), 189  
 write() (Intel method), 131  
 write() (Intel32e method), 133  
 write() (IntelPAE method), 136  
 write() (KDDEBUGGER\_DATA64 method), 383  
 write() (KMUTANT method), 370  
 write() (KSYSTEM\_TIME method), 372  
 write() (LayerContainer method), 106  
 write() (LimeLayer method), 147  
 write() (LinearlyMappedLayer method), 150  
 write() (LIST\_ENTRY method), 373  
 write() (list\_head method), 326  
 write() (mm\_struct method), 328  
 write() (MMVAD method), 375  
 write() (MMVAD\_SHORT method), 376  
 write() (module method), 329  
 write() (mount method), 330  
 write() (OBJECT\_HEADER method), 378  
 write() (OBJECT\_SYMBOLIC\_LINK method), 379  
 write() (ObjectInterface method), 111  
 write() (PdbMSFStream method), 152  
 write() (PdbMultiStreamFormat method), 155  
 write() (Pointer method), 192  
 write() (POOL\_HEADER method), 380  
 write() (PrimitiveObject method), 193  
 write() (proc method), 349  
 write() (qstr method), 331  
 write() (queue\_entry method), 350  
 write() (RegistryHive method), 163  
 write() (SegmentedLayer method), 166  
 write() (SERVICE\_HEADER method), 395  
 write() (SERVICE\_RECORD method), 396  
 write() (sockaddr method), 351  
 write() (sockaddr\_dl method), 352  
 write() (socket method), 354  
 write() (String method), 198  
 write() (struct\_file method), 332  
 write() (StructType method), 199  
 write() (super\_block method), 334  
 write() (task\_struct method), 335  
 write() (TranslationLayerInterface method), 109  
 write() (UNICODE\_STRING method), 382  
 write() (UnionType method), 200  
 write() (vfsmount method), 336  
 write() (vm\_area\_struct method), 338  
 write() (vm\_map\_entry method), 355  
 write() (vm\_map\_object method), 356  
 write() (VmwareLayer method), 168  
 write() (vnode method), 357  
 write() (Void method), 201  
 write() (WindowsCrashDump32Layer method), 128  
 write() (WindowsIntel method), 138  
 write() (WindowsIntel32e method), 140  
 write() (WindowsIntelPAE method), 142  
 write() (WindowsMixin method), 145

## Z

zfill() (Bytes method), 179  
 zfill() (HexBytes method), 316  
 zfill() (String method), 198