Volatility Documentation

Release 3.0.0$_{alpha1}$

Volatility Foundation

Oct 10, 2019

CONTENTS

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:

CHAPTER

ONE

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.

CHAPTER

TWO

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 hexidecimal 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 attribtues 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.

CHAPTER

THREE

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 whereever 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.

CHAPTER

FOUR

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 by 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

CHAPTER

FIVE

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.

CHAPTER

SIX

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 rendererd 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.

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 fourht (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/la
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 eith each other.

class Context
Bases: *volatility.framework.interfaces.context.ContextInterface*

Maintains the context within which to construct objects.

The context object is the main method of carrying around state that's been constructed for the purposes of investigating memory. It contains a symbol_space of all the symbols that can be accessed by plugins using the context. It also contains the memory made up of data and translation layers, and it contains a factory method for creating new objects.

Other context objects can be constructed as long as they support the *ContextInterface*. This is the primary context object to be used in the volatility framework. It maintains the

Initializes the context.

add_layer(*layer*)
Adds a named translation layer to the context.

Parameters **layer** (*DataLayerInterface*) – The layer to be added to the memory

Raises ***volatility.framework.exceptions.LayerException*** – if the layer is already present, or has unmet dependencies

Return type None

clone()
Produce a clone of the context (and configuration), allowing modifications to be made without affecting any mutable objects in the original.

Memory constraints may become an issue for this function depending on how much is actually stored in the context

Return type *ContextInterface*

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

Return type *HierarchicalDict*

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

Return type *LayerContainer*

module(*module_name*, *layer_name*, *offset*, *native_layer_name=None*, *size=None*)
Constructs a new os-independent module.

Parameters

- **module_name** (str) – The name of the module
- **layer_name** (str) – The layer within the context in which the module exists
- **offset** (int) – The offset at which the module exists in the layer
- **native_layer_name** (Optional[str]) – The default native layer for objects constructed by the module
- **size** (Optional[int]) – The size, in bytes, that the module occupys from offset location within the layer named layer_name

Return type *ModuleInterface*

object(*object_type*, *layer_name*, *offset*, *native_layer_name=None*, ***arguments*)
Object factory, takes a context, symbol, offset and optional layername.

Looks up the layername in the context, finds the object template based on the symbol, and constructs an object using the object template on the layer at the offset.

Parameters

- **object_type** (Union[str, *Template*]) – The name (or template) of the symbol type on which to construct the object. If this is a name, it should contain an explicit table name.
- **layer_name** (str) – The name of the layer on which to construct the object
- **offset** (int) – The offset within the layer at which the data used to create the object lives
- **native_layer_name** (Optional[str]) – The name of the layer the object references (for pointers) if different to layer_name

Return type *ObjectInterface*

Returns A fully constructed object

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

Return type *SymbolSpaceInterface*

class Module(*context*, *module_name*, *layer_name*, *offset*, *symbol_table_name=None*, *native_layer_name=None*)
Bases: *volatility.framework.interfaces.context.ModuleInterface*

Constructs a new os-independent module.

Parameters

- **context** (*ContextInterface*) – The context within which this module will exist
- **module_name** (str) – The name of the module
- **layer_name** (str) – The layer within the context in which the module exists
- **offset** (int) – The offset at which the module exists in the layer
- **symbol_table_name** (Optional[str]) – The name of an associated symbol table

- **native_layer_name** (Optional[str]) – The default native layer for objects constructed by the module

property context
Context that the module uses.

Return type *ContextInterface*

get_enumeration(*name*)
Returns an enumeration from the module.

Return type *Template*

get_symbol(*name*)
Returns a symbol from the module.

Return type *SymbolInterface*

get_type(*name*)
Returns a type from the module.

Return type *Template*

has_enumeration(*name*)
Determines whether an enumeration is present in the module.

Return type bool

has_symbol(*name*)
Determines whether a symbol is present in the module.

Return type bool

has_type(*name*)
Determines whether a type is present in the module.

Return type bool

property layer_name
Layer name in which the Module resides.

Return type str

property name
The name of the constructed module.

Return type str

object(*object_type*, *offset=None*, *native_layer_name=None*, *absolute=False*, ***kwargs*)
Returns an object created using the symbol_table_name and layer_name of the Module.

Parameters

- **object_type** (str) – Name of the type/enumeration (within the module) to construct
- **offset** (Optional[int]) – The location of the object, ignored when symbol_type is SYMBOL
- **native_layer_name** (Optional[str]) – Name of the layer in which constructed objects are made (for pointers)
- **absolute** (bool) – whether the type's offset is absolute within memory or relative to the module

Return type *ObjectInterface*

object_from_symbol(*symbol_name*, *native_layer_name=None*, *absolute=False*, ***kwargs*)
Returns an object based on a specific symbol (containing type and offset information) and the layer_name of the Module. This will throw a ValueError if the symbol does not contain an associated type, or if the symbol name is invalid. It will throw a SymbolError if the symbol cannot be found.

Parameters

- **symbol_name** (str) – Name of the symbol (within the module) to construct
- **native_layer_name** (Optional[str]) – Name of the layer in which constructed objects are made (for pointers)
- **absolute** (bool) – whether the symbol's address is absolute or relative to the module

Return type *ObjectInterface*

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

Return type int

class ModuleCollection(*modules*)
Bases: object

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

deduplicate()
Returns a new deduplicated ModuleCollection featuring no repeated modules (based on data hash)

All 0 sized modules will have identical hashes and are therefore included in the deduplicated version

Return type *ModuleCollection*

get_module_symbols_by_absolute_location(*offset*, *size=0*)
Returns a tuple of (module_name, list_of_symbol_names) for each module, where symbols live at the absolute offset in memory provided.

Return type Iterable[Tuple[str, List[str]]]

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

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

class SizedModule(*context*, *module_name*, *layer_name*, *offset*, *size*, *symbol_table_name=None*, *native_layer_name=None*)
Bases: *volatility.framework.contexts.Module*

Constructs a new os-independent module.

Parameters

- **context** (*ContextInterface*) – The context within which this module will exist
- **module_name** (str) – The name of the module
- **layer_name** (str) – The layer within the context in which the module exists
- **offset** (int) – The offset at which the module exists in the layer
- **symbol_table_name** (Optional[str]) – The name of an associated symbol table
- **native_layer_name** (Optional[str]) – The default native layer for objects constructed by the module

property context
Context that the module uses.

Return type *ContextInterface*

get_enumeration(*name*)
Returns an enumeration from the module.

Return type *Template*

get_symbol(*name*)
Returns a symbol from the module.

Return type *SymbolInterface*

get_symbols_by_absolute_location(*offset*, *size=0*)
Returns the symbols within this module that live at the specified absolute offset provided.

Return type List[str]

get_type(*name*)
Returns a type from the module.

Return type *Template*

has_enumeration(*name*)
Determines whether an enumeration is present in the module.

Return type bool

has_symbol(*name*)
Determines whether a symbol is present in the module.

Return type bool

has_type(*name*)
Determines whether a type is present in the module.

Return type bool

property hash
Hashes the module for equality checks.

The mapping should be sorted and should be quicker than reading the data We turn it into JSON to make a common string and use a quick hash, because collissions 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 statisfying
- **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 callback 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, automagic 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 HierachicalDict 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 occupys 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 being 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 LayerContainer

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

read(*layer*, *offset*, *length*, *pad=False*)
Reads from a particular layer at offset for length bytes.

Returns 'bytes' not 'str'

Parameters

- **layer** (str) – The name of the layer to read from
- **offset** (int) – Where to begin reading within the layer
- **length** (int) – How many bytes to read from the layer
- **pad** (bool) – Whether to raise exceptions or return null bytes when errors occur

Return type bytes

Returns The result of reading from the requested layer

values() → an object providing a view on D's values

write(*layer*, *offset*, *data*)
Writes to a particular layer at offset for length bytes.

Return type None

class ScannerInterface
Bases: 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 potental 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 (eithe 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 transfered 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.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 callback 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 callback 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 => b implies that a + c => 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 openning 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.automagic.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'xb9x01xef'.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 potental 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 is 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
~volatilit y.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 VolTem plateProxy)

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
~volatilit y.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 Elfs(*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.lsmod module

A module containing a collection of plugins that produce data typically found in Linux's /proc file system.

class Lsmod(*context*, *config_path*, *progress_callback=None*)
Bases: *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.lsmod module

A module containing a collection of plugins that produce data typically found in Mac's lsmod command.

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.

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 indicies 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.vadyarascan 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 descendent is a child, grandchild, etc of node.

max_depth()
Returns the maximum depth of the tree.

static path_depth(*node*)
Returns the path depth of a particular node.

Return type int

path_sep = '|'

populate(*function=None*, *initial_accumulator=None*)
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 endianess 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 hexidecimal 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'xb9x01xef'.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 potental 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*, *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 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.

retreive_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 indentifier 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

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

V

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z